2017

Network Partitioning in Distributed Agent-Based Models

Antoniya Petkova
University of Central Florida

NETWORK PARTITIONING IN DISTRIBUTED AGENT-BASED MODELS

by

ANTONIYA PETKOVA
B.S. Bethune-Cookman University, 2006
M.S. University of Central Florida, 2010

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2017

Major Professor: Narsingh Deo

ABSTRACT

Agent-Based Models (ABMs) are an emerging simulation paradigm for modeling complex systems, comprised of autonomous, possibly heterogeneous, interacting agents. The utility of ABMs lies in their ability to represent such complex systems as self-organizing networks of agents. Modeling and understanding the behavior of complex systems usually occurs at large and representative scales, and often obtaining and visualizing of simulation results in real-time is critical.

The real-time requirement necessitates the use of in-memory computing, as it is difficult and challenging to handle the latency and unpredictability of disk accesses. Combining this observation with the scale requirement emphasizes the need to use parallel and distributed computing platforms, such as MPI-enabled CPU clusters. Consequently, the agent population must be "partitioned" across different CPUs in a cluster. Further, the typically high volume of interactions among agents can quickly become a significant bottleneck for real-time or large-scale simulations. The problem is exacerbated if the underlying ABM network is dynamic and the inter-process communication evolves over the course of the simulation. Therefore, it is critical to develop topology-aware partitioning mechanisms to support such large simulations.

In this dissertation, we demonstrate that distributed agent-based model simulations benefit from the use of graph partitioning algorithms that involve a local, neighborhood-based perspective. Such methods do not rely on global accesses to the network and thus are more scalable. In addition, we propose two partitioning schemes that consider the bottom-up individual-centric nature of agent-based modeling. The first technique utilizes label-propagation community detection to partition the dynamic agent network of

an ABM. We propose a latency-hiding, seamless integration of community detection in the dynamics of a distributed ABM. To achieve this integration, we exploit the similarity in the process flow patterns of a label-propagation community-detection algorithm and self-organizing ABMs.

In the second partitioning scheme, we apply a combination of the Guided Local Search (GLS) and Fast Local Search (FLS) metaheuristics in the context of graph partitioning. The main driving principle of GLS is the dynamic modification of the objective function to escape local optima. The algorithm augments the objective of a local search, thereby transforming the landscape structure and escaping a local optimum. FLS is a local search heuristic algorithm that is aimed at reducing the search space of the main search algorithm. It breaks down the space into sub-neighborhoods such that inactive sub-neighborhoods are removed from the search process. The combination of GLS and FLS allowed us to design a graph partitioning algorithm that is both scalable and sensitive to the inherent modularity of real-world networks.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

ABM agent-based model, a computational model that represents autonomous entities and the interactions between them (see Section 2.2). 1

Catastrophe Theory a branch of mathematics concerned with systems displaying sudden shifts in behavior arising from small changes of a variable or a force applied on the system. 75

community structure a grouping of the nodes in a network into (potentially overlapping) sets such that each set of nodes is densely connected internally and sparsely connected to other sets. 15

complex system a system made of numerous autonomous elements that interact in a nonlinear fashion. Global system behavior is irreducible to the simple behaviors of the elements. 5

D-MASON is the distributed version of the MASON multi-agent platform. 13

EcoLab an agent-based modeling software package that supports parallel and distributed simulations. 13

emergence defines the development of higher-level systemic phenomena that cannot be traced back causally to microscopic rules and behaviors (see Section 2.2.3). 6

FLAME an agent-based modeling toolkit that provides support for parallelization. 13

Ja-Be-Ja a fully distributed graph partitioning algorithm (see Section 5.3.4). 65

Jostle a parallel multilevel graph partitioning software package. 15

KaHIP Karlsruhe High Quality Partitioning suite (see Section 5.3.3). 15

***k*-means clustering** an unsupervised learning algorithm that classifies a given set of data points into k clusters such that each point belongs to the cluster with the nearest centroid. 16

Kronecker a model for generating networks with realistic features (see Section 3.4.3.1). 30, 56, 67

LFR Lancichinetti–Fortunato–Radicchi, a benchmark tool that generates networks with power-law distributions (see Section 3.4.3). 32, 56

LPA Label-Propagation Algorithm, a community-detection algorithm in which each node carries the label of the community it belongs to. At each iteration, the nodes adopt the most popular labels in their neighborhoods (see Section 3.2.2). 16

MASON an agent-based modeling toolkit that supports sequential simulations. It was designed as a smaller and faster alternative to Repast. 13

METIS a software package used for serial partitioning of graphs and meshes (see Section 5.3.1). 15

NetLogo an agent-based modeling toolkit that supports sequential simulations. 13

ParMETIS an MPI-based library that extends METIS and offers parallelized methods for graph partitioning. 15

PuLP a distributed graph partitioning algorithm based on the label-propagation paradigm (LPA). 16

Repast an agent-based modeling toolkit. Repast Symphony supports sequential simulations while Repast HPC supports parallel and distributed simulations. Repast offers no network partitioning and load-balancing mechanisms (see Section 2.3). 13

Scotch software toolkit for sequential and parallel graph partitioning, static mapping and clustering, sequential mesh and hypergraph partitioning, and sequential and parallel sparse matrix block ordering. 15

self-organization a dynamic process by which complex macroscopic behaviors arise spontaneously from simple generative rules and behaviors. 6

SIR Susceptible-Infected-Recovered, an epidemiological model of the spread of disease in a population of agents (see Section 3.2.1). 20

SpaDES a modeling environment specializing in spatially explicit models, including raster-based, event-based, and agent-based models. It does not provide native network-distribution functionality. 13

Swarm an agent-based modeling toolkit that does not provide native network-distribution functionality. 13

CHAPTER 1: INTRODUCTION

Several application domains, including sociology [1, 2], biology [3, 4], defense [5], and economics [6, 7], utilize systems that have two common characteristics: (1) they are complex and large in scale – made up of a huge number of autonomous entities and (2) the entities themselves are dynamic, i.e., the interactions and relationships between them change over time. For example, in a country-wide epidemiological scenario, each person could be an entity, and the interactions between people (which relate to the spread of disease) are dynamic, i.e., they change over time. Formally, these systems can be modeled as graphs, such that entities and their interactions are represented as vertices and edges, respectively [8, 9]. Moreover, in such systems, there are two aspects to consider – the "local" aspect, at micro-scale, such as the spread of disease from person to person, and the "global" aspect, at macro-scale, which is a view of how the disease is spreading overall within a geographical region.

Agent-based models (ABMs) are an emerging simulation paradigm for modeling such complex systems, comprised of autonomous, possibly heterogeneous, interacting agents [10]. The ability of ABMs to define complex systems in terms of entities (agents) and their dynamics at micro-scale facilitates high fidelity of the modeled elements and processes. However, in many complex systems, the global system behavior emerges from the, sometimes seemingly chaotic, ensemble of individual actions of the autonomous agents (akin to the schooling or flocking behavior of fish or birds) [7, 11]. Such emergent behaviors often become evident only at very large scales (e.g. building of cathedral mounds by termites) [12]. Further, if the goal is to track, understand, predict, and act on the spread of disease, for example, its simulation by agent-based modeling has to take place in real time. Therefore, two clear requirements emerge: (1) modeling and understanding the be-

havior of such systems at *large and representative* scales, and (2) obtaining and visualizing the simulation results in *real time*.

The real-time requirement necessitates the use of in-memory computing, as it is challenging to handle the latency and unpredictability of disk accesses. Combining this observation with the large scale requirement emphasizes the need to use parallel and distributed computing platforms, such as MPI-enabled CPU clusters. Consequently, the agent population has to be "partitioned" across different CPUs in a cluster. Further, the typically high volume of interactions among agents can quickly become a significant bottleneck for real-time simulations. The problem is exacerbated if the underlying ABM network is dynamic and the inter-process communication evolves over the course of the simulation. Therefore, it is critical to develop topology-aware partitioning mechanisms to support such large simulations.

Existing work relevant to the problem at hand can be summarized loosely into the following three groupings:

1. ABM simulation environments developed for domain experts: Such environments focus on ease of use as opposed to performance. Historically, these have supported sequential simulations [13, 14] and, in recent years, some distributed environments have been developed [15, 16]. However, they do not offer adequate partitioning mechanisms.

2. Traditional graph-partitioning techniques: While it is intuitive and natural to model the ABM network using a graph (vertices being agents and edges being their interactions), it is inefficient to take a traditional graph-partitioning technique (such as a multilevel algorithm) and call it repeatedly as a subroutine for every change in the evolving graph.

3. Community-detection algorithms: These algorithms exploit a common characteristic in real-world graphs – community structure – an inherent organization of the entities into densely connected groups that are loosely connected with each other [17]. While community-detection algorithms are gaining traction, their applicability to large and dynamic graphs in the context of distributed computing remains largely unexplored.

In this dissertation, we demonstrate that distributed agent-based model simulations benefit from the use of graph-partitioning algorithms that involve a local, neighborhood-based perspective. Such methods do not rely on global accesses to the network and thus are more scalable. In addition, we propose two partitioning schemes that consider the bottom-up individual-centric nature of agent-based modeling. The first technique utilizes label-propagation community detection to partition the dynamic agent network of an ABM. We propose a latency-hiding, seamless integration of community detection in the dynamics of a distributed ABM. To achieve this integration, we exploit the similarity in the process flow patterns of a label-propagation community-detection algorithm and self-organizing ABMs.

In the second partitioning scheme, we apply a combination of the Guided Local Search (GLS) and Fast Local Search (FLS) metaheuristics in the context of graph partitioning. The main driving principle of GLS is the dynamic modification of the objective function to escape local optima. The algorithm augments the objective of a local search, thereby transforming the landscape structure and escaping a local optimum. FLS is a local search heuristic algorithm that is aimed at reducing the search space of the main search algorithm. It breaks down the space into sub-neighborhoods such that inactive sub-neighborhoods are removed from the search process. The combination of GLS and

FLS allowed us to design a graph-partitioning algorithm that is both scalable and sensitive to the inherent modularity of real-world networks.

The rest of the dissertation is organized as follows. In Chapter 2, we present relevant background information. In Chapter 3, we lay the groundwork for a "topology-aware" partitioner, based on community detection. In Chapter 4, we propose another graph partitioning algorithm with strong locality, GF-Part, based on the combination of two metaheuristics. In Chapters 5, we introduce the design for a performance study that evaluates the role of graph partitioning in agent-based model simulations. In Chapter 6, we discuss future research that focuses on using Catastrophe Theory concepts to facilitate effective network redistribution. We present our concluding remarks and future directions in Chapter 7.

CHAPTER 2: BACKGROUND

In this chapter, we provide relevant background information, which should reveal the context of the problem and motivate its study.

2.1 Complex Systems

Complex systems are made of numerous, possibly diverse, autonomous elements that interact with each other, typically in a nonlinear fashion. Complex systems may form and evolve through self-organization, facilitating the development of emergent behavior at macroscopic scales [18]. As Sayama points out, it is helpful to understand what complex systems are not. They are not collections of independent entities, such as an ideal gas, nor are they collections of strongly coupled bodies [18]. The first example falls into the category of "problems of disorganized complexity" and is handled by statistics, and the latter – "problems of simplicity"– where the system can be described with a small set of variables. Both terms were coined by the mathematician and systems scientist Warren Weaver [19]. Complex systems fill in the gap between these two extremes and form the category of "problems of organized complexity" [18, 19]. However, they are not engineered systems, such as an airplane, put together according to a blueprint [20]. Instead, complex systems emerge from dynamic processes with a high degree of stochasticity influenced by dependency and bias [21].

While the complex dynamics of the system affects the expression of its structure properties, the structure itself critically influences the dynamics of the whole [21–23]. The structures of complex systems are emerging from the evolving web of interconnections

between the elements of the system. Therefore, dynamic networks can be used effectively to express this evolving topology. They capture structural aspects of complex systems by encoding elements as nodes and the interactions between them as links. Despite their diversity, these networks can be derived by a common set of structural and evolutional properties, such as degree distribution, distance, betweenness, community structure, etc. Based on these features, we recognize several classes of complex networks. One such class is that of the Erdős-Rényi random networks, where the links are assumed to exist completely independently of each other [24]. In contrast, in real-world networks, the existence of links is typically influenced by nonindependent processes and bias. Another prominent class is that of small-world networks proposed by Watts & Strogatz [25]. In these networks, the average distance (shortest path) between two nodes can be orders of magnitude smaller than the number of nodes making up the network [26]. Another ubiquitous category is that of the scale-free networks [27]. These are networks, whose degree distributions follow a power law. This property is a result of two generative mechanisms: 1) the network forms by the continuous addition of new nodes, and 2) the new nodes attach preferentially to nodes with high degrees.

These common network properties give rise or are directly related to the common properties shared by the systems themselves. Despite the large variety of complex systems, there is a series of unifying principles and statistical properties that are shared by most of them [28]. One such feature, emergence, concerns the manifestation and the nontrivial relationship of the rest of the system's properties at different scales [18]. Emergence denotes the development of higher-level phenomena that cannot be traced back causally to microscopic rules and behaviors. Another important principle of complex systems is self-organization, a dynamic process by which complex macroscopic behaviors arise spontaneously from simple generative rules and behaviors [26]. Global patterns and order

emerge from the interactions of the entities (with each other or the environment) without explicit instructions. Emergence and self-organization are tightly related to another property, nonlinearity, which states that the emergent effects are rarely proportional to causes, and what happens locally in a system does not translate directly to a cumulative effect. In other words, the outputs of a system are not produced by a linear combination of the inputs [18]. As a result of these properties, complex systems are counterintuitive; cause and effect are distant in time and space [29]. Therefore, we need modeling tools that allow us to capture the properties and processes of the system at different scales, from the ground up.

2.2 Agent-Based Models

Agent-based modeling is a rule-based, discrete-event and -time modeling paradigm that facilitates the simulation and study of complex systems. These systems consist of a large number of interacting autonomous entities [30]. The entities may be heterogeneous in attributes and behavior. Most importantly, their interactions produce an emergent effect that cannot be derived directly from the effects of the individual behaviors [7]. Agent-based models (ABMs) attempt to bridge the semantic gap between complex systems and their computational models by providing a bottom-up approach to building the systems' representations and dynamics. Specifically, ABMs facilitate the study of real phenomena by modeling them on a microscopic level – the constituent discrete entities; their internal states; individual, possibly heterogeneous, behaviors; and interactions with each other. A typical agent-based model has three elements [11]:

1. A set of agents, including their attributes and behaviors.

2. A set of agent relationships forming an underlying topology of connectedness that defines how and with whom agents interact.

3. The agents' environment (optional) representing the virtual world, in which these agents exist. This world may be completely neutral and not affect the agents' behaviors and interactions; or it may be a complex model that causes agents to react [31].

2.2.1 Agents

The modeled entities are known as agents. Even though there are no established standards for modeling agents, there is an assumed modularity associated with their representation. This modularity allows for a certain degree of freedom in specifying the agents' individual attributes, states, and behaviors. This wide range of possibilities, in turn, leads to a greater fidelity of the model. Each agent is self-contained, uniquely identifiable, and has a state that changes over time [11, 32]. In general, agents are assumed to have the following additional traits [33]:

- autonomy – the capability to operate without external input and human intervention;
- social ability – the capability to interact with other agents in the system;
- reactivity – the capability to react to changes or stimuli from the modeled environment;
- intelligence (optional) – the ability to learn;
- adaptability (optional) – the ability to adjust behavior and state according to a changing environment;

An ABM can have several different types of agents, each characterized by its own set of attributes, behaviors, and internal states. The complexity of the system depends very much on the complexity and number of agents. An ABM could represent both systems with a huge number of very simple agents and systems with a small number of very complex agents, or any configuration in between.

2.2.2 Topologies

A central concern of ABMs is modeling agent interactions – how agents are connected and what mechanisms drive their interactions. ABMs offer three different topologies for representing connectivity in complex systems [32]. In some cases, the structure of agent interactions can be represented as simple regular lattices (grids), as in a Cellular Automata model. Each agent moves from cell to cell on the grid and interacts with other agents only if they are located in the cells immediately surrounding that of the agent. ABMs also accommodate movements and interactions of entities on a continuum. This kind of topology allows the agents to exist and move in an unstructured multi-dimensional environment. The relationships between agents depend on their spheres of influence (SOIs) – a region of the space that circumscribes each agent. Any agents that fall in each others' SOIs are considered neighbors.

A large number of complex systems have interaction structures that are too complex to be represented by a simple topological model. In such cases, the network topology is used to model the social connectivity of the agent population as a graph, where the agents and their interactions are represented as vertices and edges, respectively. The edges in the network do not have to reflect physical proximity. Oftentimes, the graph is dynamic, i.e. its structure may evolve as part of the dynamics of the agent interactions or behaviors.

If necessary, one could combine two or even all three topologies in the same model. It is worth noting that the unifying factor in all topology variants is that agents' interactions are localized. Each agent communicates only with others in its neighborhood. There is no global source of information; interaction is mainly confined in small subsets of the agent population and thus the global system processes are very much regulated and dependent on this modularity of the interaction topology.

2.2.3 Local and Emergent Behavior

As already noted, agent-based modeling is a decentralized, individual-centric modeling paradigm. The modeled system lacks a centrally organized global behavior; there is no central authority that controls or influences the behavior of the individuals in an attempt to optimize the global system performance [11]. Agents receive only local information – either through interactions with neighbors, or through a localized interaction with the environment – and use it to regulate their own behaviors and decisions. On the other hand, the evolving topology of connectedness is usually tightly interleaved with the individual behavior of the agents.

The behaviors of the modeled entities are autonomous and usually heterogeneous. They are governed by locally-constrained rules and are characterized by intrinsic stochasticity. The localized interactions between agents generate collective systemic dynamics that are unrelated and irreducible to the individual behaviors. In many natural systems without central authority, collective (swarm) intelligence emerges as a result of the interaction of agents with seemingly simple behaviors [10]. In an agent-based model, the individual behaviors may be encoded with equations, or they may be specified by decision rules, such as if-then statements.

Each agent goes through changes in its state that can be attributed both to its individual behavior and the influence of others. In many cases the emergent behavior may arise as a result of the broken balance between individual behavior and peer influence. In such situations, the effect of the exchange propagates, takes dominance over autonomous decision-making, and results in a large-scale self-organized global behavior [34]. Allowing an emergent behavior to develop from low-level actions and interactions provides insight into the causes and circumstances of its occurrence. However, such behavior is usually only evident when the system is modeled at a relevant scale – a significant requirement that ABMs are able to accommodate.

In addition to macroscopic and microscopic occurrences, many complex systems also exhibit behavioral and communication patterns on a mesoscopic level, i.e. the tendency of the agent population to organize into communities. Similarly to the system-level behavior, these structures might not appear to be directly associated with the individual behaviors of the agents. Thus, ABMs might be a useful tool for gaining insight into the coupling of the system behavior and the evolving structure of complex systems under study.

2.2.4 Environment

The environment is the virtual world in which agents operate. It could be an active participant in the dynamics of the modeled system, or completely neutral and non-reactive [31]. The environment is useful not only because it corresponds to the role of the environment in the real system and thus increases the fidelity of the model, but also because it might facilitate the monitoring of the agents. In addition, it could be used to buffer the agent communication in order to achieve synchronicity. Common environments include geo-

graphical spaces, such as cities, or biological structures like the mammalian cell.

2.2.5 ABMs vs Analytical Models

While mathematical modeling is an essential tool in every scientist's toolbox, because of its well-formulated approach to representing and computing different phenomena under study, it is not especially effective when these phenomena are complex. In fact, some systems are so complex that they are mathematically intractable – formulating an analytical model to describe them is close to impossible [35]. In such situations, simulation models could be useful in providing insight into the structure and dynamics of the modeled system. They can be particularly useful for hypothesis testing and scenario analysis [36].

ABMs, in particular, can handle and truthfully represent many of the aspects of complex systems that render their mathematical models intractable. Some of these complexity-inducing factors include 1) a massive number of autonomous entities; 2) heterogeneity of the entities; 3) interaction between entities; and 4) randomness. In many cases, the heterogeneity of the parts, their ability to interact, and the randomness in their behaviors are irreducible to a mean behavior or a set of attributes and have to be modeled explicitly. ABMs offer a direct correspondence between the modeled entities and their real-world counterparts. The generative approach of ABMs provides for significant explanatory power – the system may be built from the bottom up in as much measurable detail as necessary. That is, a modeler can design both the agents and their environment with arbitrary complexity and truthfulness to capture the modeled system as observed [37]. Thus, scientists have the chance to look at the full dynamics of the systems at all levels and study the coupling of the process flow and the structural evolution of the interaction topology. Finally, ABMs lend themselves to parallelization well and thus can simulate

systems comprised of hundreds of million and even billions of agents.

On the other hand, analytical models assume a reductionist perspective w.r.t. the modeled system – a population-level view, where the individual attributes and behaviors are reduced to a mean representative. Equation-based models, such as System Dynamics, do not represent the entities of the system directly. Instead, such models transform entities into probability distributions and focus on aggregate quantities such as the projected number of infected humans at a given time-step in an epidemiological model [38]. The assumption is that all agents are the same or belong to a few categories of uniform character and behavior. In addition, the classic approach to modeling natural phenomena, such as epidemics, assumes uniform mixing. That is, the individuals in a population are assumed to come in contact with equal probability, independent of their locations [21]. However, it is well known that the processes of disease propagation and other natural and social phenomena are localized and depend on the nature of contact. As a result, this approach to system representation leads to a certain degree of dissociation between the natural structure and behavior of the system and its model.

2.3 ABM Simulation Platforms

Several modeling environments have been developed in recent years to facilitate the simulation of agent-based models. Toolkits like Repast [13], NetLogo [14], Swarm [39], SpaDES [40], and MASON [41] are designed to handle sequential simulations and thus offer inadequate computational scalability for large-scale models [11]. Despite recent developments of high performance solutions like EcoLab [42], Repast HPC [15], FLAME [43], and D-MASON, [16] offering support for parallel and distributed simulations, features like network partitioning (especially, topology-centric) and load balancing are still miss-

ing [44].

Repast HPC is considered the most comprehensive ABM simulation environment. In its core, the tool implements the fundamental building blocks of Repast Symphony (the current version of Repast for serial simulations) but adapts them to work on parallel and distributed environments. One of the main purposes of this modeling environment is ease-of-use and flexibility, that is, to be accessible for users from different domains. While this objective might have been fulfilled to a great extent, fundamental performance-optimization features, such as topology-aware model partitioning and load-balancing, are still missing. DMASON, the distributed multi-agent simulation kit based on MASON, is conceptually similar to Repast HPC, but offers rudimentary partitioning functionality. Partitioning of the model is decided by the user in advance and is not sustained in the course of the simulation. The lack of continuous topology-aware partitioning support could lead to deterioration in the communication overhead and increased load imbalance in distributed ABM simulations with dynamic contact networks.

2.4 Use of Traditional Graph-Partitioning Algorithms

The graph-partitioning problem is a well-known NP-hard problem that concerns the partitioning of a given graph into a predefined number of components, such that the number of edges connecting the components is minimized [45]. A variant of this problem, uniform (or balanced) graph partitioning – where the components have to be similar in size, is especially relevant in the context of parallel and distributed computing. The goal of such algorithms is to minimize the inter-process communication overhead while maintaining near uniform load distribution across the processes. Previous studies that focused on the problem of uniform graph partitioning produced solutions in the form of heuristic-based

or approximation algorithms. Most graph-partitioning algorithms that are effective on large graphs are based on the multilevel graph-partitioning paradigm [46–48]. Some of these are offered in packages such as METIS and ParMETIS [49], Jostle [50], Scotch [51], and KaHIP [52]. Most of the traditional graph-partitioning algorithms are either inadequate or not efficient enough in the context of massive self-organizing complex networks for one or more of the following reasons:

- Most traditional approaches rely on a centralized, omniscient view of the network, assuming cheap access to the entire network. They typically involve numerous global operations on the network and therefore might not be a natural fit for the agent- or vertex-centric nature of self-organizing ABMs.
- The multilevel graph-partitioning approach is memory-intensive and not efficient on massive complex networks. The goal of most traditional graph-partitioning algorithms is accuracy, at the cost of efficiency.
- The aforementioned algorithms have to be executed as stand-alone procedures when applied to large distributed simulations of self-organizing complex networks; they are hard to integrate into the dynamics of the networks. Such an approach incurs additional overhead and negatively impacts the execution time of the simulation.

2.5 Use of Community-Detection Algorithms

Community structure is an intrinsic property of many complex networks and can be used to reveal structural and behavioral characteristics of modeled complex systems from different domains [28, 53]. Currently, only several other simulation approaches utilize community detection in the parallel and distributed execution of ABMs. In [54], Hou et al.

present a community-based partitioning scheme for distributing the parallel simulation of ABMs (of social networks). The proposed method involves the use of the Infomap community detection algorithm to discover communities in a large-scale social network [55]. Based on the power-law distribution of the resultant communities, a weighted network is formed in which the communities are abstracted as vertices. A k-way partitioning algorithm is used to map the communities onto k compute nodes such that the load is well balanced and the inter-node communication is minimized. Unfortunately, the algorithm is designed to be applied only on static social networks, such as the network of collaborations between movie actors. It applies community detection only once – at the start of the simulation.

Wang et al. propose a cluster-based partitioning scheme for the distributed execution of agent-based crowd simulations [2]. The method models crowd dynamics as a dynamic network in which each agent is represented as a node. Each agent is associated with its own area of interest (AOI), which acts as its communication range. The network-partitioning algorithm is based on k-means clustering, which is used for the detection of clusters of agents. The algorithm assigns each AOI to a compute node and tries to minimize the overlap (communication overhead) between AOIs occupying different processors. Cluster-detection is applied periodically to a snapshot of the dynamic network and agents are redistributed. The authors report reduction in the communication bottleneck; however, they ignore any overhead incurred by the cluster detection and the redistribution of the network.

In [56], Slota et al. introduce a distributed graph-partitioning method, PuLP, based on the label-propagation paradigm [57]. The partitioner uses a label-propagation algorithm (LPA) to partition small-world networks on distributed platforms while trying to preserve locality and minimize memory usage. PuLP works in three stages. The first stage

initializes the network to be partitioned. The second and third stages alternate a label-propagation-based load balancing step and a refinement step that further improves an optimization objective. While the authors report notable performance improvement over METIS, PuLP was designed for simulations of processes on static networks or to provide an initial partition for simulations of dynamic networks.

CHAPTER 3: USING LABEL-PROPAGATION COMMUNITY DETECTION TO DISTRIBUTE AGENT-BASED MODEL SIMULATIONS

3.1 Overview

In this chapter, we show how a community detection technique can be successfully applied to partition the dynamic network (describing the agent population) of an ABM. To the best of our knowledge, we are the first to propose a latency-hiding, seamless integration of community detection in the dynamics of a distributed ABM. This strategy lowers the communication overhead incurred by the constant migration of agents in the network. Specifically, our contributions are as follows:

- We propose a latency-hiding, seamless integration of community detection in the dynamics of a distributed ABM. We exploit the similarity in the process flow patterns of a label-propagation community-detection algorithm and self-organizing ABMs to achieve the integration.
- We demonstrate how periodically adjusting the mapping between the communities and the processes, without having to run a standalone community-detection procedure beforehand, can be efficient.
- We present a framework for evaluating the frequency of network remapping in a distributed platform.
- We implement and compare three different schemes that apply LPA to an ABM and we demonstrate that our approach achieves a speedup of up to eight times over the baseline (a random distribution of the agents across processors). We implement the

algorithms both in a standalone ABM simulation in C and as part of Repast HPC to showcase the usefulness of the proposed approach.

The rest of the work is organized as follows. In Section 3.2, we describe the agent-based model. In Sections 3.3, we describe our approach and present the respective algorithms. We discuss our experimental study and the results thereof, along with analysis of our findings, in Sections 3.4 and 3.5. We conclude the chapter and discuss our future work in Section 3.6.

3.2 Simulation Model

In this study, our ABM of choice is the SIR (Susceptible-Infected-Recovered) epidemiological model [58]. We represent the contact network of the ABM as a dynamic undirected graph. Let $G = (V, E)$ be an undirected, unweighted graph with a vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and an edge set $E = \{e_1, e_2, \ldots, e_m\} \subseteq (V \times V)$, representing the set of agents and the interactions among them, respectively. Then, a *dynamic graph* $\mathcal{G}$ is a series of time-dependent snapshots $\mathcal{G} = \{G^0, G^1, \ldots, G^{t_{max}}\}$, where $G^t = (V^t, E^t)$ is a static graph generated at time $0 \leq t \leq t_{max}$. Let $N(v_i) = \{v_j \in V : \langle v_i, v_j \rangle \in E\}$ denote the neighborhood set of vertex v_i. Thus, each agent communicates only with others in its neighborhood. By migrating to a different part of the network, agents change their neighborhoods, which is encoded in the snapshots of the dynamic graph. Furthermore, we simulate the spread of disease and community detection on the network via a message-passing approach. Therefore, each vertex has two attributes l_i and h_i, representing its community membership and health status, respectively. The roles of the attributes will be clarified further in subsequent sections.

3.2.1 The SIR Model

Our agent-based model of choice is the SIR (Susceptible-Infected-Recovered) epidemiological model. The SIR ABM simulates the spread of disease in a population of agents such that each agent falls in one of three categories: susceptible, infected, or recovered (see Figure 3.1). We simulate the spread of disease through the exchange of messages between connected agents. Each message carries the health condition of the sender (i.e., h_i) as payload (Algorithm 1). The disease propagation process is governed by two parameters: the rate of infection and the rate of recovery. We assume that the population size remains constant; that is, we do not include an agent birth and death mechanism. Each agent updates its status depending on its own health condition and that of its neighbors. We assume that each agent sends (receives) a message, symbolizing social contact and disease transmission, to (from) all of its neighbors at each iteration of the simulation. The stopping condition of the simulation is the complete eradication of the disease, i.e., when the number of infectious agents reaches zero, or when a predefined number of iterations is reached. The recovery process of each agent is stochastic in nature. Specifically, for each infected agent the algorithm draws a random number from a uniform distribution between 0 and 100. If the number is less than the recovery rate, the agent's state changes to 'recovered' (and therefore, immune). If not, the agent remains infected. Similarly, for all susceptible agents, the algorithm draws a random number from a uniform distribution between 0 and 100. If the number is less than the rate of infection, the agent's state changes to 'infected'.

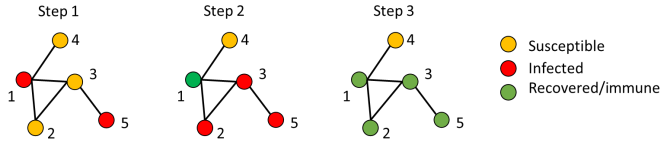

Figure 3.1: The SIR ABM, in which agents exchange messages to simulate social contact and the transmission of disease. In step 1, agents 1 and 5 become infected. In step 2, the disease is transmitted to agents 2 and 3. Meanwhile, agent 1 has recovered. In step 3, all infected agents have recovered.

Algorithm 1 Standalone SIR_SIM Message-Passing Mechanism

```
1: procedure RUN_SIR_SIM(G, N(G), t)
2:     for all v_i ∈ V(G) do
3:         send h_i[t − 1] to ∀v_j ∈ N(v_i)
4:         receive h_j[t − 1] from ∀v_j ∈ N(v_i)
5:         update h_i[t]                          ▷ update health status of v_i
6:     end for
7: end procedure
```

3.2.2 The Label Propagation Algorithm

Only a fraction of the community detection algorithms are efficient enough to be used in the processing of large complex networks. A few attain near-linear time complexity. One of the most prominent methods in that category is the label-propagation algorithm (LPA), introduced by Raghavan et al [57]. For the purposes of our study, we adopt an instance of the LPA community-detection class of algorithms, called Max-LPA [59]. The motivation behind our choice is that both LPA and most self-organizing ABMs are similarly decentralized, individual-centric in nature. LPA, like many self-organizing complex networks, achieves order through decentralized interactions between individual elements.

Algorithm 2 Standalone Max_LPA Message-Passing Mechanism

1: **procedure** RUN_MAX_LPA(G, $N(G)$, t)
2: **for all** $v_i \in V(G)$ **do**
3: send $l_i[t-1]$ to $\forall v_j \in N(v_i)$
4: receive $l_j[t-1]$ from $\forall v_j \in N(v_i)$
5: update $l_i[t]$ ▷ update community membership of v_i
6: **end for**
7: **end procedure**

In addition, Max-LPA, specifically, has the following beneficial properties: 1) no randomness besides the initial label assignment, which renders the algorithm deterministic (and stable); 2) tie resolution is deterministic – always in favor of the larger (numerical) label; 3) the algorithm lends itself well to parallelization due to the synchronous nature of the label updates; and 4) the algorithm demonstrates fast convergence.

Every node in the network is assigned a unique label uniformly and independently at random. At each time step, every node sends its label to all of its adjacent nodes. After receiving labels from all neighbors, each node updates its label with the most frequently occurring label in its neighborhood (Algorithm 2). Ties are broken by selecting the label with the highest numerical value (Figure 3.2).

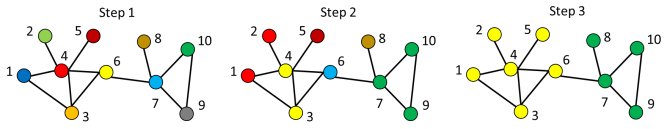

Figure 3.2: Progression of Max-LPA. The colors of the nodes represent their community labels. In step 1, all nodes have unique labels. In step 2, ties are resolved by assigning labels with higher numerical value (based on agent id). For instance, we can see that nodes 1 and 2 become red, because they acquire the label of node 4. In step 3, all nodes have been separated into 2 communities.

Algorithm 3 Applying Max_LPA only once, at the beginning of simulation

```
1: procedure STATIC_LPA(G, N(G))
2:     t ← 0
3:     initialize the network
4:     run_MAX_LPA()
5:     while true do
6:         redistribute network on processors
7:         run_SIR_SIM()
8:         evolve network
9:         t++;
10:    end while
11: end procedure
```

3.3 Proposed Solution

In this section, we examine three different optimization scenarios for applying Max-LPA on an ABM network: Initial Application, Periodic Application, and Integrated Application. The applicability of each of these scenarios can vary depending on the nature of the ABM, and we examine the tradeoffs between them. Further, we introduce an optimization in Integrated Application that reduces overhead and improves efficiency. We also present a framework for evaluating the frequency of network remapping in a distributed platform.

3.3.1 Initial Application of Max-LPA

In this approach, we apply Max-LPA on the network only once, before starting the ABM simulation (refer to Algorithm 3). We use the detected communities to distribute the agent population across the processors. Following the partitioning of the network, we

let the ABM simulation progress until completion. This approach is similar to [54, 56] and is only appropriate for static networks. This is because, in a dynamic network, the initial community structure changes as the simulation progresses and agents migrate to different communities.

3.3.2 Periodic Application of Max-LPA

The following partitioning scheme provides an improvement over the aforementioned single application of community detection (refer to Algorithm 4). We perform an initial LPA community detection on the network before starting the ABM simulation. In addition, we apply community detection and re-distribute the network at regular intervals to prevent the degradation of community quality over time. The label propagation is executed for several iterations initially, and only one iteration for every subsequent simulation step of the ABM.

Algorithm 4 Applying Max_LPA at every iteration of the SIR simulation

```
1: procedure DYNAMIC_LPA(G, N(G))
2:     t ← 0
3:     initialize the network
4:     run_MAX_LPA()
5:     while true do
6:         redistribute network on processors
7:         run_SIR_SIM()
8:         evolve network
9:         run_MAX_LPA()
10:        t++;
11:    end while
12: end procedure
```

Algorithm 5 Integrated LPA_SIR Message-Passing Mechanism

1: **procedure** RUN_LPA_SIR(G, $N(G)$, t)
2: **for all** $v_i \in V(G)$ **do**
3: send $l_i[t-1]$ and $h_i[t-1]$ to $\forall v_j \in N(v_i)$
4: receive $l_j[t-1]$ and $h_j[t-1]$ from $\forall v_j \in N(v_i)$
5: update $l_i[t]$ and $h_i[t]$
6: **end for**
7: **end procedure**

3.3.3 Integrated Application of Max-LPA

Our approach is based on the idea that LPA and most self-organizing ABMs share the same communication pattern. In particular, in both cases dynamic processes are running on the network, simulating the concept of flow. In both cases the flow of the processes is realized through the communication of each agent with its neighbors. However, especially when the size of the graph increases, communication between processors can generate prohibitive overhead, mandating the need for optimization. The key insight behind our proposed scheme is that the label-propagation mechanism can piggyback on the communication between agents in the ABM to transmit the label (community membership) information (see Algorithm 5). Here we integrate the MAX-LPA algorithm in the ABM simulation. LPA relies solely on the structure of the network to drive its process. We utilize the common communication pattern between the two algorithms and combine both types of messages into a single one (refer to Algorithm 6). This is the most efficient scheme of the three because it subsumes the overhead of community detection.

Algorithm 6 Integrating Max_LPA into the SIR simulation

```
1: procedure INTEGRATED_LPA(G, N(G))
2:     t ← 0
3:     initialize the network
4:     while true do
5:         t++;
6:         evolve network
7:         run_LPA_SIR
8:         redistribute network on processors
9:     end while
10: end procedure
```

3.3.4 ABM Redistribution

As mentioned above, our integrated Max-LPA algorithm runs continuously as part of the ABM dynamics. However, in order to benefit from the detected community structure, the simulation has to remap it by migrating the affected agents between processes. This process of dynamic remapping often incurs considerable run-time overhead and thus raises the concern of how often to redistribute the workload so that the overhead of this operation does not outweigh its benefit. In this section, we present our effort to answer this question in the form of a remapping decision policy.

Cost Model: The cost of ABM simulation increases with time after a remapping of the ABM network, as a result of the dynamic nature of the graph and agents migrating between communities (see Figure 3.3). In this work, we assume that the rate of agent migration is known, thus we can estimate empirically the cost of an ABM simulation step at time t given that the last remapping was completed s steps ago. We represent this as $f_t(s)$. The cost to re-organize the agents by community also increases with time, because more agents have migrated communities since the last remapping and must be moved.

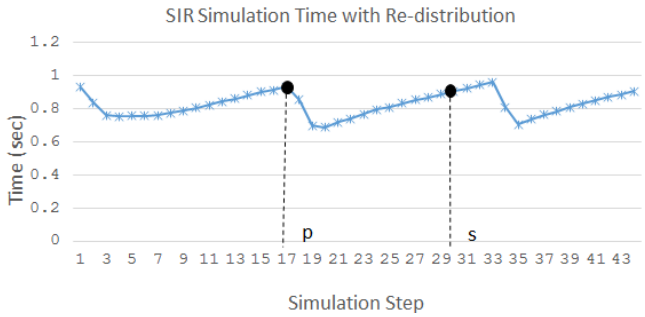

Figure 3.3: The SIR model run-time deteriorates gradually after every redistribution step. At time step step 17 (p) we perform a redistribution of the graph. Later, at time step 30 (s), the run-time of the SIR model has already deteriorated due to agents migrating.

Similarly, the cost of remapping at time t, given that the last remapping was performed s steps earlier, is represented by $B_t(s)$. Our remapping decision policy attempts to minimize the ABM simulation time by weighing the costs of remapping at a given time step against the costs of continuing the simulation without remapping. To understand this trade-off, we first formulate the cost of delaying the remapping and compare it to the cost of performing the remapping at a given time step. The parameters used in the formulation are listed in Table 3.1. At a given time step t, our policy evaluates the future cost of the model simulation, provided a network remapping was performed s steps ago.

Table 3.1: Parameters Used in Remapping Decision Policy

parameter	description
t	simulation time step
s	time steps since last remapping
R	remaining simulation time
$f_t(s)$	simulation cost if remapping occurred s steps ago
$B_t(s)$	remapping cost if last remapping occurred s steps ago
n	total number of simulation steps
$C_k(n)$	average cost of remapping at every k steps

If that future cost exceeds the combined cost of remapping at t and executing the remaining simulation, then remapping should proceed at t. In other words, it is more cost-effective to redistribute the network communities on the processors at the current time step, than let the simulation proceed as is. Formally, we express this notion as follows:

$$\sum_{t}^{R+t} f_t(s) > B_t(s) + \sum_{t}^{R+t} f_t(0),$$

where $\sum_{t}^{R+t} f_t(s)$ is the cost of the remaining R simulation steps, provided remapping were executed s steps ago but omitted at the current step t. $B_t(s)$ signifies the cost of remapping at t, given that the last remapping was completed s steps in the past. The last term designates the cost of the remaining simulation, if we remap the network at t.

With this insight in mind, we seek to determine the optimal frequency of network redistribution by examining the cost of remapping at different interval values. We define the total cost *C* to run a simulation with remapping at every k steps over the course of n simulation steps:

$$C_k(n) = \sum_{t=1}^{n} f_t(t \bmod k) + \sum_{t=1}^{n/k} B_t(t \bmod k).$$

Thus, we aim to find a k such that the value of *C* is minimized. The model considers both the cost of delaying the remapping and the overhead of remapping at an interval of k steps.

The values of the parameters in the cost model depend on the configuration of the simulation environment. We performed an empirical evaluation of the model for both our stand-alone tool and RepastHPC and derived a remapping frequency optimal for our experimental setup.

3.4 Experimental Setup

We implemented the simulation of the SIR model with the three partitioning schemes, presented in Section 3.3, both as a standalone simulation in C and as part of the Repast HPC platform.

3.4.1 Standalone Tool

The standalone simulation in C utilizes OpenMPI 1.8.1-1.el6.x86_64 message-passing protocol. All experiments were executed on a CPU-cluster with the following configuration: 10 compute nodes, Dual-Socket Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.7 GHz 30MB LLC, 128 GB DDR3 memory, 500GB SATA Disk and 10 Gigabit Ethernet.

3.4.2 Repast HPC

We implemented our LPA-based partitioning algorithm in Repast HPC – a modeling environment that enables the highly-parallelized and distributed simulations of massive ABMs [15]. Repast HPC was developed at the Argonne National Laboratory based on the core concepts and principles of Repast Symphony – a toolkit mainly aimed at facilitating sequential simulations of ABMs. It was developed in C++ using MPI (enhanced by the Boost library) for parallel operations. Some of the inherited features include representing agents as objects, scheduling events via a dynamic discrete-event scheduler, model parametrization via property files, and data collection. To facilitate parallelization, Repast HPC incorporates features like cross-process communication, synchronization, and agent migration.

Every process is responsible for its local agents. In addition, each process has a number of borrowed agents, i.e., copies of agents that reside on other processes but share edges with agents local to the given process. Sharing of agent copies simplifies the process of agent communication by providing local agents easy access to adjacent agents that are hosted by a different process. Any updates of the original agents require cross-process synchronization with the remotely-stored copies. The synchronization ensures that the pan-process model is simulated as a coherent whole. Repast HPC accomplishes the migration, copying, updating, and synchronization of agents between processes using a 'package pattern' [60]. The package encapsulates the state of an agent (or an edge) so that it can be recreated on a different process. Processes exchange packages over MPI using the Boost serialization and MPI libraries.

We ran the Repast HPC experiments on a CPU-cluster with the following configuration: 3,578 total compute cores (Intel(R) Xeon(R) CPU E5-2680 @ 2.7 GHz), over 7.5TB of RAM, and a 20/40Gb Infiniband interconnect. We scheduled our jobs to utilize 128 processors from this cluster with 3GB of memory per processor.

3.4.3 Benchmarks

We used two different benchmark toolkits to generate synthetic test graphs.

3.4.3.1 Kronecker Network Generation Model

One of the graph generators we used was the Kronecker network generation model, part of the Stanford Network Analysis Platform (SNAP). SNAP is a general purpose, high performance system for analysis and manipulation of large networks [61]. The graph

generator, KronGen is empirically proven to effectively model the structure of real networks and is mathematically tractable [62]. The generator has been proven to obey main static network patterns that have appeared in the literature as well as temporal evolution patterns [63].

In addition, SNAP includes KronFit, a linear time, scalable algorithm for fitting the Kronecker graph generation model to real networks [64]. One of the benefits of this feature is that it allows us to generate extrapolations of smaller real networks. These extrapolations are larger versions of the original graphs, evolved via a set of temporal growth patterns.

We used the Kronecker generator to extrapolate a large version of a small real network. Specifically, we used KronFit to extract the essential features of the *dolphins social network* graph [65]. We then used these features to grow dolphins with KronGen (see Table 3.2). In addition, we used KronGen to generate a synthetic graph of over one million vertices. The characteristics of the graph are very similar to those of numerous real-world networks.

Table 3.2: Parameters Used for Graph Generation

parameter	description	G_1	G_2	Kron1M	KronDolphin
N	number of agents	32,768	1,048,576	1,048,576	65,536
k	average degree	15	20	–	–
maxk	maximum degree	50	200	–	–
μ	mixing parameter	0.1-0.5	0.1-0.5	–	–
minc	min community size	20	10	–	–
maxc	max community size	50	200	–	–
–	agent migration percentage	5%	1-4%	–	–

3.4.3.2 *The Lancichinetti–Fortunato–Radicchi Benchmark Tool*

Most of the test networks were created using the Lancichinetti–Fortunato–Radicchi (LFR) benchmark tool [66], as described in Table 3.2. The tool generates networks with power-law distributions. It allows the user to control parameters such as the mixing parameter (the percentage of neighbors of a node that do not belong to the same community as that node), average node degree, and community size.

All simulation runs we executed lasted 50 iterations. In each iteration, a percentage of the agents were selected at random to change communities, thereby modeling the dynamic nature of the networks. The properties (listed in Table 3.2) of the initial version of each network created by LFR were preserved while migrating agents.

3.5 Performance Evaluation

We compare the performance of the three schemes described in Section 3.3 and a baseline approach which does not use community detection. This approach distributes the agent population randomly across the CPUs. We present the results in the following subsections.

3.5.1 *Standalone Simulation – Results and Discussion*

Here, we present and discuss the results obtained by testing the three partitioners in the standalone simulation of the SIR model.

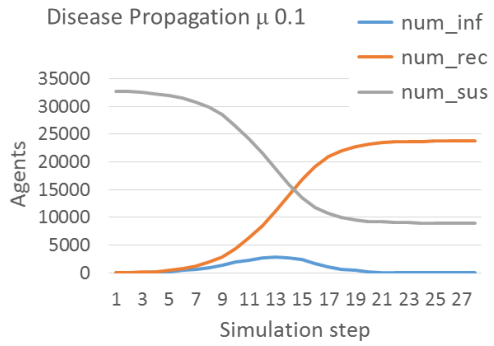

Figure 3.4: Disease propagation on network G_1 with $\mu = 0.1$.

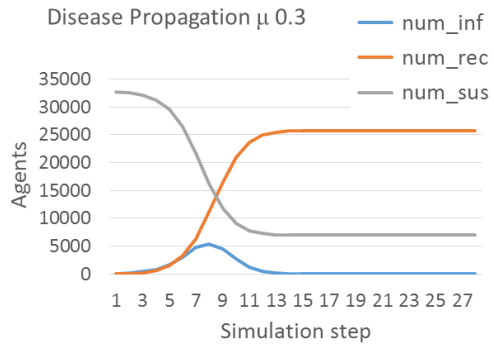

Figure 3.5: Disease propagation on network G_1 with $\mu = 0.3$.

3.5.1.1 Effect of Mixing Parameter

In our experiments, one of the parameters we varied was μ, the mixing parameter. It specifies the fraction of a vertex degree attributed to edges connecting that vertex with vertices outside of its community. Thus, a higher value of μ indicates more connectivity between the communities in the graph. We used the following values of μ in our experiments: 0.1, 0.2, 0.3, 0.4, and 0.5. Figures 3.4 and 3.5 show the disease propagation for values of 0.1 and 0.3, respectively.

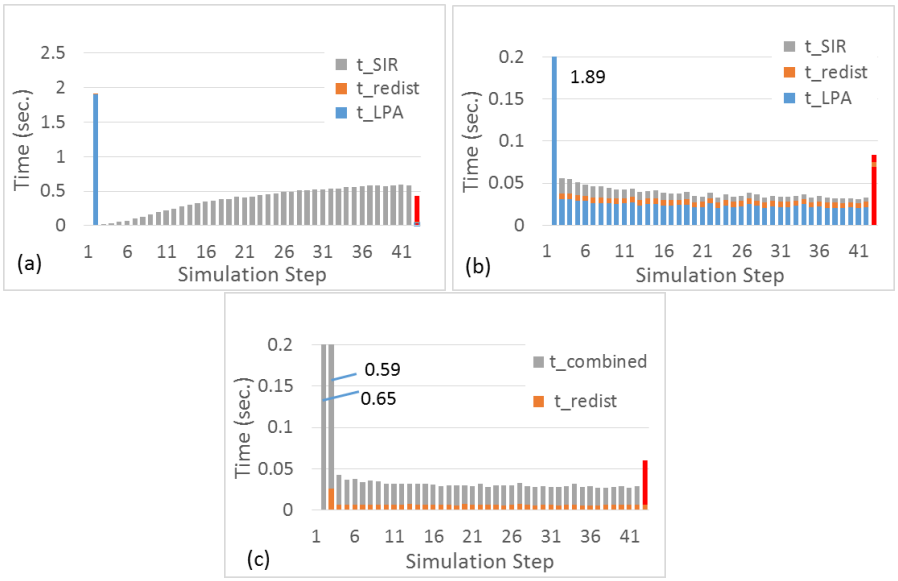

Figure 3.6: Simulation details for the three schemes for network G_1 with $\mu = 0.1$: (a) *Initial-LPA*, (b) *Separate-LPA*, (c) *Integrated-LPA* show the execution time for different steps of the 3 schemes respectively. Note that the scale on the y-axis for (a) is different from that of (b) and (c).

The plots show how the numbers of infected, recovered, and susceptible agents varied over the course of the simulation.It can be observed that the slopes of the curves are steeper when $\mu = 0.3$ as compared with $\mu = 0.1$; this supports the intuition that the processes of spread and recovery evolve faster when the communities are more tightly connected (higher value of μ) [58].

3.5.1.2 Run-time Overhead

One of the most significant performance results we collected reveals the difference in the execution times of the three partitioning routines. Figure 3.6 shows the simulation times for $\mu = 0.1$. Figure 3.6(a) depicts high initial overhead caused by the community detection procedure (shown in blue). However, subsequent steps of the SIR ABM simulation are

very efficient, in comparison, because of the agents being distributed on the computing cluster according to community membership. However, as the simulation progresses, the graph changes with the migration of agents across communities. As a result, the communities detected in the initial step become increasingly inaccurate. This is reflected in the increase in the time spent per simulation step (because of MPI communication). Note that the majority of the simulation time (more than 90%) was spent on communication, as evident in all experiments.

Figure 3.6(b) shows the execution time of the second scheme, in which we call the Max_LPA procedure at every step of the SIR simulation. Note that the y-axis is at a different scale for this experiment. We achieved a consistent and low simulation time throughout the run, because each ABM simulation step was performed after a call to the Max_LPA procedure, and thus benefited from the freshly redistributed (according to community membership) network. In the plot, we distinguish between the performance results of Max_LPA and the SIR model by assigning them different colors.

It can be observed that the execution times, demonstrated by both, are approximately equal. This is because, as in the previous experiment, the dominant factor is the communication overhead – and the communication patterns are identical in both procedures. If we consider the average time taken per iteration (last bar in red), we see that it is approximately 4.2x better than in the previous case.

Figure 3.6(c) shows performance results obtained from the third scheme. Here, instead of completing the community detection before the SIR simulation, we overlapped the two by integrating the label propagation routine into the message-passing mechanism of the SIR model. We observe that the first two simulation steps took longer, until the community structure was refined sufficiently; subsequent steps yielded significant improvement.

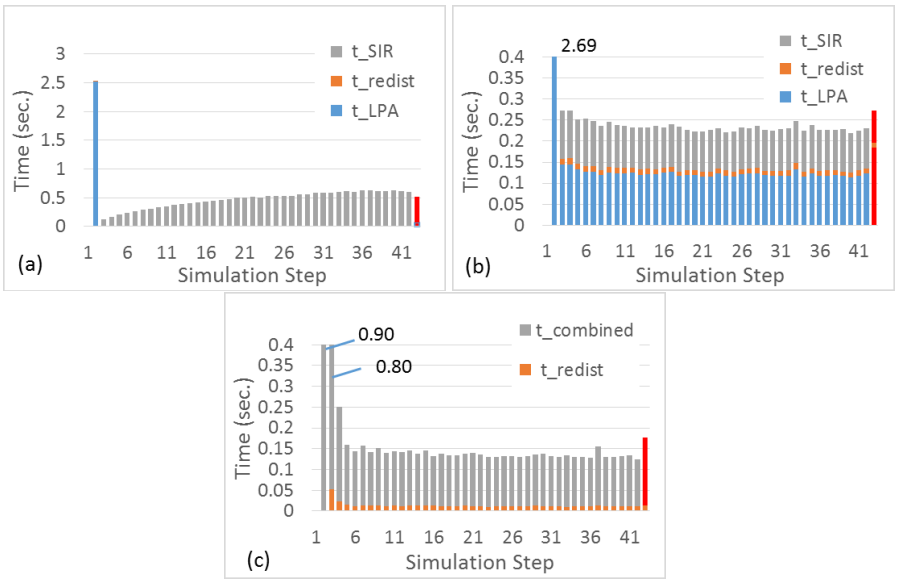

Figure 3.7: Simulation details for the three schemes for network G_1 with $\mu = 0.3$: (a) *Initial-LPA*, (b) *Separate-LPA*, (c) *Integrated-LPA* show the execution time for different steps of the 3 schemes respectively. The scale on the y-axis for (a) is different from that of (b) and (c).

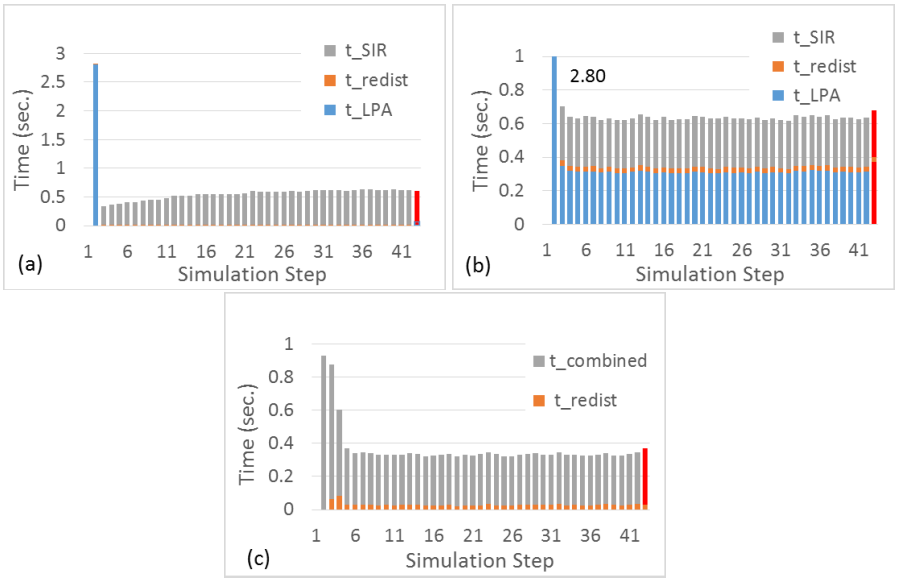

Figure 3.8: Simulation details for the three schemes for network G_1 with $\mu = 0.5$: (a) *Initial-LPA*, (b) *Separate-LPA*, (c) *Integrated-LPA* show the execution time for different steps of the 3 schemes respectively. The scale on the y-axis for (a) is different from that of (b) and (c).

We observe that this is the best performing scheme of the three. Label $t_{combined}$ in the graph designates the time to perform both community detection and the SIR model, combined. This is only 5% greater than the time to run the SIR model simulation only (and can be understood as the overhead of integrating the community detection into the SIR model simulation). The average execution time per iteration is 8x lower than in the first scheme, and 1.67X lower than in the second scheme.

Figures 3.7 and 3.8 show the simulation times for higher values of μ = 0.3 and 0.5, respectively. We can make the following observations based on these plots: (1) as the value of μ increases, the execution time per simulation step also increases. This is because the communities within the network become less pronounced with the increase of μ, and therefore the benefit of community detection is reduced. (2) we observe that due to the very low overhead of community detection in the third scheme, we still see significant benefit even at higher values of μ. For example, with μ = 0.5, the third scheme is 1.65x better than both other schemes.

3.5.1.3 A Side-by-side Comparison

Figure 3.9 shows a comparison of the total simulation time for all three schemes, for different values of μ, along with that of a baseline algorithm with no community detection. Again, we observe the average simulation time increases with the increase in μ. Note that the first and the second schemes perform worse than the baseline for high values of μ. This is because the overhead of community detection in these schemes is high; with high μ values, the gains attributed to improved MPI communication become less than the overhead. The third scheme is always beneficial, because of its low community detection overhead. For μ 0.1, our combined scheme outperforms the baseline by 8x.

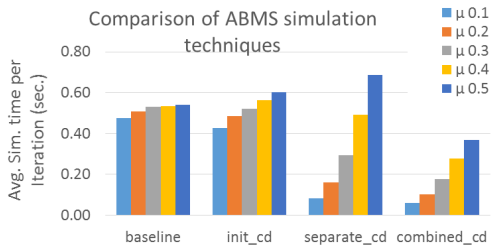

Figure 3.9: Comparison of the three schemes applied on network G_1 for different values of μ.

3.5.2 Repast HPC – Results and Discussion

In this section, we examine the run-time overhead of the SIR model for different network partitioning scenarios. Figure 3.10 compares the performance of our baseline (no redistribution by community) against the three proposed schemes, i.e. *Initial-LPA*, *Separate-LPA*, and *Integrated-LPA*. These results are quite revealing in several ways. First, as in Section 3.5.1, we can report significant performance benefits in the execution of the ABM simulation as a result of applying community detection. On average, the execution time per iteration of the baseline case is 3.5 times higher than that of our integrated-LPA approach (not accounting for the overhead of redistribution by community).

Second, we observe that in Repast HPC our separate and integrated approaches incur significantly higher costs for remapping the network by community than in our standalone tool. In our separate and integrated schemes we used a redistribution interval of $k = 8$ for illustrative purposes only. Our results demonstrated that unless the simulation was long enough to amortize the cost of periodic remapping, the initial-LPA approach proved more cost-effective.

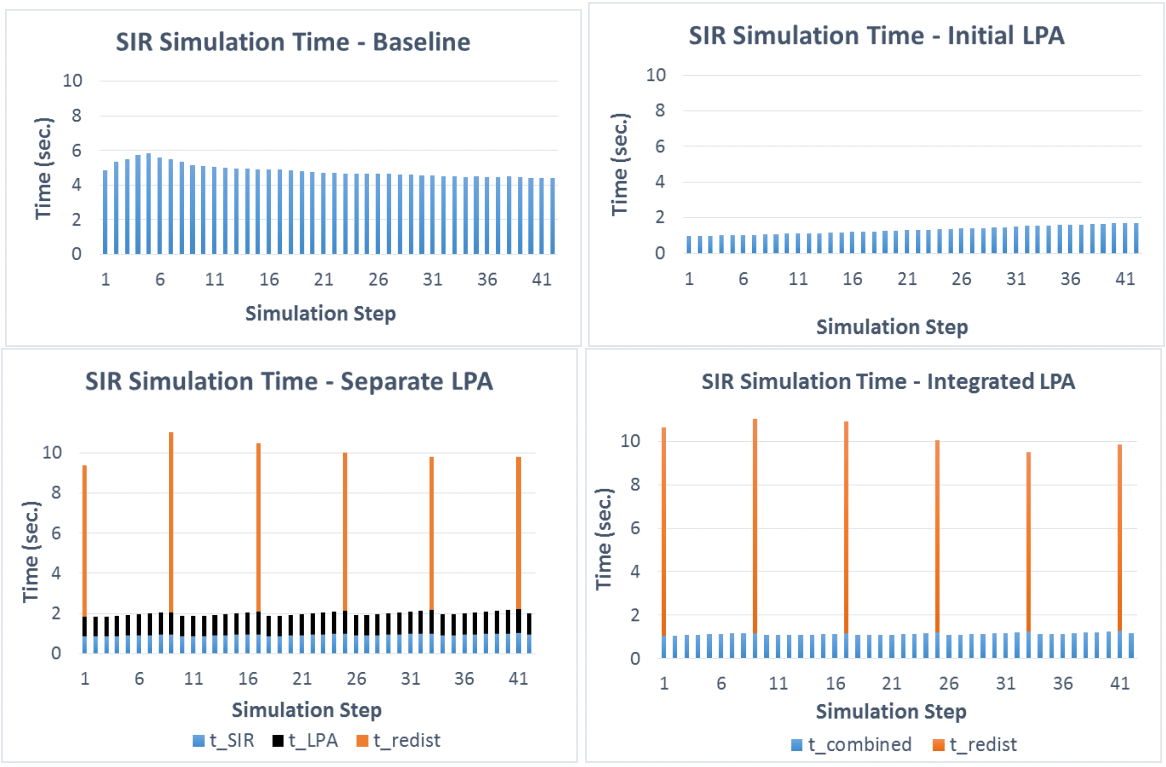

Figure 3.10: Simulation details for the three schemes for network G_2 with $\mu = 0.1$: (a) *baseline (upper left corner)*, (b) *initial application of LPA (upper right corner)*, (c) *separate, dynamic application of LPA (lower left corner)*, and (d) *integrated LPA (lower right corner)* show the execution time of the four schemes, respectively, over the course of the simulation.

The significantly higher cost for agent migration in Repast HPC is due to the inefficient implementation of that functionality. The cost of the cross-process migration of large numbers of agents is potentially prohibitive unless the model simulation is long, in which case the initial redistribution of the network amortizes over the course of the simulation. Such a scenario also justifies any periodic migration of agents to reflect the evolved communities in the network.

Table 3.3: Comparison between LPA-based Partitioner and ParMetis

Dataset	LPA Edge-cut	ParMetis Edge-cut	LPA Time (s)	ParMetis Time (s)
Kron1M	5,205,266	4,619,657	**13.65**	49.48
KronDolphin	**1,791,146**	2,555,043	5.75	2.23
LFR-1M-mu0.1	3,256,628	1,871,847	51.81	5.22

3.5.3 Comparison with State-of-the-Art

In this section, we compare the performance of the LPA-based partitioner with that of ParMetis [48,67] in terms of both edge-cut and execution time. For this set of experiments we used the simulation in Repast HPC and we focused only on the initial partitioning phase. The results shown in Table 3.3 demonstrate that in most cases ParMetis outperforms the LPA-based partitioner. This is most probably due to the fact that the local search in our algorithm settles in a local optimum. The results show that our partitioner is faster in the case of Kron1M, however, even if we let it run longer, it is unlikely that it would reach a better solution, because the algorithm lacks the ability to diversify the search. We address this issue in the next chapter.

In the case of KronDolphin, which is a very dense graph, the LPA-based partitioner outperforms Repast HPC in terms of edge-cut. In contrast, LFR-1M-mu0.1 contains predominantly small communities with sizes up to 200 vertices. Even though they are mostly well formed (due to $\mu = 0.1$) and the community detection algorithm probably detects them correctly, it is not likely that they would be distributed effectively across multiple processors. This issue is due to the lack of diversification of the local search and a strategy to lead it to a more optimal global solution.

It is worth noting that even though ParMetis seems to outperform the LPA-based parti-

tioner in many cases, our algorithm will keep refining the results in the course of the SIR simulation with negligible addition cost. Community detection runs continuously and does not have to be called as a standalone routine periodically. Calling ParMetis to periodically partition the dynamic graph of the SIR simulation would be costly in comparison to running our algorithm seamlessly. In addition, a significant part of the execution time of the LPA-based partitioner is spent on synchronization, as a result of the inefficient implementation of these routines in Repast HPC.

3.6 Conclusion and Future Work

In this chapter, we studied the use of label-propagation community detection to partition the simulation of ABMs in distributed environments. We studied three schemes for combining LPA with a SIR model simulation. The novelty lies in the seamless integration of the process flow of LPA and the SIR model. We demonstrated that our approach achieved a speedup of up to eight times for networks in which we varied the quality of the community structure. Unresolved issues suggest the benefits of extending this research to refine the partitioning strategy and study its behavior on much larger synthetic networks with various topological characteristics, as well as on real-world networks. In addition, future research should focus on developing a more efficient remapping method with a corresponding cost model and a load-balancing scheme. In the next chapter, we propose a partitioning algorithm that guides the local search in escaping local optima to reach better global solutions.

CHAPTER 4: GF-PART – FAST GRAPH-PARTITIONING USING GUIDED LOCAL SEARCH

4.1 Introduction

Graph partitioning is a fundamental combinatorial optimization problem. A graph consists of a set of vertices connected by a set of edges such that every edge connects only two vertices. Graph partitioning refers to a process of partitioning the set of vertices into mutually disjoint sets with the objective that the number of edges connecting the sets is minimized. Searching for the optimal such partition is computationally prohibitive for large graphs due to the phenomenon combinatorial explosion (the number of combinations to be examined grows exponentially fast). Graph partitioning is an NP-hard problem, which means that exhausting the search space is unlikely to satisfy the computational demand of most modern systems or simulations. In such cases, heuristic methods are used to obtain good results in a reasonable amount of time.

Graph partitioning is increasingly relevant in practice in the context of data processing and analysis. As established in previous chapters, a wide variety of real-world data sets can be modeled as graphs. As a result of the ever-growing size of such graphs, it is necessary to exploit their inherent structural characteristics in order to partition the graphs into manageable chunks. The smaller parts can then be processed in parallel on high-performance computing platforms. Such partitioning of the data according to their natural modularity allows each vertex to be processed in the context of its neighborhood. Preserving locality is especially critical when modeling complex systems, such as the spread of disease, as agent-based models where the exchange of information is primarily local.

In this chapter, we propose a graph partitioning algorithm, GF-Part, that utilizes a combination of the Guided Local Search (GLS) and Fast Local Search (FLS) metaheuristics [45]. GLS is a general penalty-based metaheuristic that usually sits on top of other local search heuristic methods and guides them to improve their efficiency. FLS also aims to improve the efficiency of a local search but through reducing the size of the search neighborhoods.

4.1.1 Guided Local Search

Guided Local Search is a deterministic single-solution metaheuristic that is used to guide an underlying local search algorithm [68]. The main driving principle of GLS is the dynamic modification of the objective function in order to escape local optima. The algorithm modifies the features of the obtained solution, thereby transforming the landscape structure and escaping a local optimum.

In GLS (see Algorithm 7), a set of m features is associated with every solution. The features describe characteristics imposed on the solutions in the search space by the specific problem under consideration. For example, a feature in the Traveling Salesman Problem could be the link (edge) between cities (vertices) A and B in a given candidate tour (solution). For each solution $s \in S$, where S is the search space, a feature f_i is determined by the *indicator function* I_i, $i \in \{1, ..., m\}$, as follows:

$$I_i(s) = \begin{cases} 1, & \text{if } s \text{ has property } f_i, \\ 0, & \text{otherwise} \end{cases}$$

In addition, GLS associates each feature f_i with a cost c_i and a penalty p_i. Thus, when the search is trapped in local optima, solutions with certain features are penalized.

Algorithm 7 GLS

$t \leftarrow 0$
Crate an initial solution
for $i = 1$ to m **do**
    $p_i = 0$
end for
while Stopping criterion is not met **do**
    $h = g + \lambda \sum_{i=1}^{m} p_1 I_i$
    $s_t + 1 = runFLS(V, s_t, h)$; ▷ or any other local search heuristic
    for $i = 1$ to m **do**
        $util_i \leftarrow I_i(s_t + 1) \times c_i/(1 + p_i)$; ▷ compute the utility of features
    end for
    for each i such that $util_i$ is max **do**
        $p_i \leftarrow p_i + 1$
        Activate all relevant sub-neighborhoods
    end for
    $t++$;
end while
$s* \leftarrow$ best solution found with respect to objective function g
return s*

Given an objective function g, which maps every candidate solution s to a numerical value, GLS defines an augmented cost function that modifies g by penalizing the solution:

$$h(s) = g(s) + \lambda \times \sum_{i}^{m} (p_i \times I_i(s)),$$

where λ represents the relative weight of penalties with respect to the cost of the solution. Whenever the local search gets trapped into a local optimum, GLS modifies the cost function by penalizing the features that are present or unfavorable. The selection is made according to the features' utility values:

$$util_i(s) = I_i(s) \frac{c_i}{1 + p_i}.$$

Thus, if a feature is not present in the current solution, its utility value is zero. The features with highest utility are penalized by incrementing the values of their penalties.

4.1.2 Fast Local Search

Fast Local Search is a local search heuristic algorithm that is aimed at reducing the search space of the main search algorithm [69]. The current neighborhood of the search is broken down into smaller neighborhoods. FLS allows us to associate each neighborhood with a solution, thereby assisting GLS in using the solution features to guide the search locally.

Each smaller neighborhood is assigned a bit that indicates whether that neighborhood is active. The active neighborhoods are the only ones involved in the search. Initially, all of them are active. However, if a neighborhood does not contain any improving moves, it is deactivated. Inactive neighborhoods can be activated, if they are related to a neighborhood with an improving move. In the course of the search, the process dies out with fewer and fewer neighborhoods being activated until all bits turn to 0. The solution generated at that point becomes the approximate local optimum.

4.2 Problem Statement

In this section we formulate the optimization problem associated with k-way graph partitioning. We also describe the distribution model of the system.

4.2.1 *k-way Graph Partitioning*

Let $G = (V, E)$ be an undirected, unweighted graph with a vertex set V and an edge set $E \subseteq (V \times V)$, representing a set of agents and the interactions among them, respectively. In addition, let $N(v)$ denote the neighborhood set of vertex v, and let $d(v)$ denote the degree of v, that is $d(v) = |N(v)|$.

Given a graph G, a mapping $\pi : V \rightarrow \{1, 2, ..., k\}$ partitions V into k disjoint parts, $\Pi = \{\pi_1, \pi_2, \ldots, \pi_k\}$. The mapping assigns a label l, where $l \in \{1, 2, ..., k\}$, to each vertex p. Therefore, $\pi(v)$, or the shorthand π_v, refers to the label of vertex v. Thus, all vertices in a partition have the same label. Consequently, $N_v(l)$ denotes the set of neighbors of v that have label l:

$$N_v(l) = \{u \in N_v : \pi_u = l\} \tag{4.1}$$

We define the *cost* of the system as the number of edges between vertices with different labels (equivalent to edge-cut). Thus, the cost of a vertex, $cost(v)$, is the number of its neighbors with a different label than its own. We can formally define the system cost as follows:

$$C(G, \pi) = \frac{1}{2} \sum_{v \in V} (d_v - d_v(\pi_v)), \tag{4.2}$$

where $d_v(\pi_v)$ is the number of neighbors of v with label π_v. The sum is divided by two to reflect that each edge is counted once. Then, we can define the optimal partitioning Π^* of the graph as

$$\Pi^* = \arg\min_{\pi} C(G, \pi). \tag{4.3}$$

4.2.2 *Data Distribution Model*

We assume that the graph is processed in a distributed environment. Specifically, it is distributed such that each processor hosts multiple vertices. GF-Part is executed sequentially on each processor on a portion of the graph. Vertices located on different processors communicate via message passing. Local communication benefits from the shared memory on each host.

GF-Part could be modified to accommodate truly distributed scenarios, where each vertex resides on a separate processor and communicates with the rest of the network only via messages, or in a separate thread in a multi-threaded environment.

4.3 Solution

In this section, we describe GF-Part, a heuristic-based algorithm for the k-way partitioning problem. It is based on the Guided Fast Local Search (GFLS) metaheuristic which is designed to lead a local search method to an optimal solution in an efficient manner.

4.3.1 *Overview*

The goal of applying GFLS to graph partitioning is both to decrease the size of the search space and to guide the local search to reach a better global solution. The Fast Local Search aspect of the algorithm proves a suitable fit for graphs with high modularity. The method is especially suitable in the context of complex systems that have to be modeled as networks of autonomous agents. In such cases, the search space is broken into sub-neighborhoods that are then mapped to the agents and their respective neighbor-

hoods. The portion of the agent population that has to be processed decreases with every iteration of the partitioning algorithm as the FLS deactivates agents that can no longer improve their objective values. In addition, the GLS procedure allows the algorithm to reach a more optimal solution by penalizing solutions that are stuck in local optima. The interplay between GLS and the local search method leads the search through phases of intensification and diversification to a better global solution [45,70].

In GF-Part, the graph (potentially representing an agent population) is initialized by activating all vertices and randomly assigning one of k labels each, where k equals the number of partitions. During an iteration of the algorithm, every active vertex performs a local search to find a label among its neighbors such that the cost (edge-cut) of the vertex is decreased.

In this approach, the local search operator is executed sequentially among the vertices located on a single processor. However, this process can easily be extended to be fully distributed by forcing all vertices to communicate via messages or protected sharing of the local resources in a multi-threaded scenario. In this case, one can use a swap operator and allow the vertices to look for pairwise optimization of the objective function. When such a mutual improvement between two vertices is found, the vertices swap their labels. This distribution scenario preserves the initial equal load balance across multiple processors in a HPC cluster. In other words, this way one can address the *balanced k*-way graph partitioning problem.

During the local search, two scenarios are possible – the vertex either finds an improving solution or it does not. In the former case, the vertex updates its objective value and the utility values of its features, provided any of them are present in the new solution. Subsequently, if any features of a vertex are found to have the highest utility, they are

penalized, thereby improving the chances of the algorithm escaping a local optimum. When a vertex finds an improving solution, it remains active and activates all nonactive neighbors. This action is necessary because, in problems with high locality, the structure of the search space guides local search methods towards optimal solutions [70]. Any change of the objective value of a given vertex can potentially affect the objective values of its neighbors.

In the latter case, the objective value of the vertex cannot be improved and the vertex is deactivated. An inactive vertex can be reactivated in one of two cases: 1) a neighbor finds an improving solution (as previously explained), or 2) one or more of the features of the vertex are found to have maximal values. The rationale behind the second scenario is to give the vertex a chance to look for an improving solution after its objective function has been augmented.

4.3.2 Representation

One of the key aspects of building a heuristic solution is determining the representation of the solution space and the corresponding local (neighborhood) search function. Vectors are common representations (genotypes) for graphs as they represent ordered lists of decision variables that can be mapped to the vertices of the graph. In addition, the vector representation allows the use of standard search operators based on the Hamming metric, because the difference between two solutions can be calculated as the number of different decision variables [70].

4.3.3 Local Search

Many strategies can be applied in the search of a better local solution. The strategy we employ in this study is best improvement (BI), also known as steepest descent. However, any heuristic selection method such as first improvement (FI), random selection, or any other search-space reducing tactic can be used instead. In the best improvement strategy, the vertex adopts the label that improves its cost function the most. The search in the neighborhood is exhaustive and thus may be time-consuming for large neighborhoods. However, this issue is not the focus of this study and can be remedied with a number of other search-space reducing selection strategies. Furthermore, the FLS phase of GF-Part reduces the global search space by deactivating non-improving vertices.

The first improvement strategy allows the search to stop after finding the first improving solution. That way, FI may cover only a portion of a neighborhood. Both BI and FI can be used in combination with a random search method to diversify the search process.

During the local search phase of the algorithm, each active vertex v scans its neighborhood for an improving solution to its objective function. Specifically, the vertex aims to minimize its local edge-cut, i.e. $cost(v)$, and looks for a label that would satisfy this objective. Decreasing the edge-cut is equivalent to increasing the number of neighbors that share a label with v. The search stops when all candidate neighbors yield solutions worse than the current. A vertex v selects vertex u's label only if $d_v(\pi_u) > d_v(\pi_v)$. When ties occur, preference is given to the label of a neighbor local to the given vertex to decrease the number of vertices involved in the redistribution phase. If an improving solution is found, the vertex activates all its neighbors. It then determines which features are present in the new solution and computes their new utility values.

4.3.4 *Features and Augmented Objective Function*

Combining GLS and FLS allows us to associate solution features to sub-neighborhoods. That way, we can relate the features to the moves each vertex makes and observe how these moves affect both the features and the generated solutions [69]. GF-Part associates three different features f_i, where $i \in \{1, 2, 3\}$, with every vertex v for every solution s. The first feature indicates whether a vertex has to move to a different processor as a result of a new label. The second feature explores whether $cost(v)$ is higher than the number of internal edges of v. Finally, the third feature keeps track of whether v has donated a label during the current iteration of the local search. The presence of a feature in a solution is given by the indicator function as described in Section 4.1.1. For example:

$$I_1(s_v) = \begin{cases} 1, & \text{if } \pi_v == \text{processor rank of } v, \\ 0, & \text{otherwise} \end{cases}$$

where s_v is the local solution of v. The features capture the properties of each solution s and then appropriately modify the objective function by adding penalties as described in Section 4.1.1.

4.3.5 *Feature Costs*

As noted in Section4.1.1, each feature f_i is associated with a cost c_i and a utility value $util_i$. The costs are used in the computation of the utility values, which on the other hand are used to decide which features should be penalized. The more a feature is penalized, the smaller its utility becomes, effectively removing the feature from future solutions (see Algorithm 7).

Features impose or strengthen constraints on the search space as they gather information about the local solutions. The cost and the number of previous penalties are the determining factors in the decision to penalize a feature in a local optimum. To escape the local optimum, GF-Part penalizes the features in s with highest utility values.

In GF-Part we keep the costs of f_1 and f_3 fixed, i.e. $c_1 = c_3 = 1$; however c_2 varies with the difference between $cost(v)$ and the number of internal edges. Specifically, f_2 is concerned with whether the current solution (associated with a specific vertex) yields a higher external degree for that vertex than internal. A positive answer to this question leads to the modification of c_2 (it is assigned the difference between the external and internal degrees of vertex v) and $util_2$.

4.3.6 The lambda Parameter

The only parameter in GF-Part is λ. It determines how much the increased penalties would impact the augmented objective function [71]. The larger the value of λ, the more aggressive the search and closer it gets to a random search. It allows the search to avoid solutions with penalized features by letting it execute large jumps in the search space. Small values of λ allow for slower and more thorough exploration of the search space. However, the search might require a high number of penalties to escape a local optimum.

4.3.7 The GF-Part Algorithm

The GF-Part algorithm combines the local search method and GFLS. Algorithm 8 presents FLS which sequentially checks whether the vertices are active. Each active vertex executes the $LocalSearch()$ procedure (see Algorithm 9) during which it can remain active or get

deactivated. The $findBestImprovement()$ searches the neighborhood of vertex v for the most improving label, and then it returns the selected neighbor, if one was found.

Algorithm 8 FLS

```
1: bitCounter ← V.size()
2: while bitCounter > 0 do
3:     for i = 1 to V.size() do
4:         if v → isActive() then
5:             v → LocalSearch()
6:             if v → isActive()! = true then
7:                 bitCounter − −;
8:             end if
9:         else bitCounter − −;
10:        end if
11:    end for
12:    i + +;
13: end while
14: s* ← best solution found with respect to objective function g
15: return s*
```

Algorithm 9 LocalSearch

```
1: procedure UPDATESOLUTION(v)
2:     neighbor ← findBestImprovement(v);
3:     if neighbor == null then
4:         v → setActive(false);
5:     else v → setLabel(neighbor → getLabel());
6:         for all f_i ∈ features do
7:             if f_i is active in s then
8:                 update util_i;
9:             end if
10:        end for
11:        update h ;
12:        v → activateNeighbors();
13: end if
14: return h;
15: end procedure
```

If no improving label is found, v deactivates itself. If v found an improving solution, it updates its own label with the newly found one. The vertex then proceeds to update any of the features that might be present in the solution. Finally, v updates its objective function and activates any neighbors that might be inactive. After a predetermined number of iterations, GF-Part uses the final solution of GFLS, i.e. the latest vertex labels, and redistributes the graph on the high-performance cluster.

4.4 Experimental Study

In this section, we present the results from our experimental study. We first focus on the performance of GF-Par with respect to four different label selection strategies. We then study the effect of λ on the edge-cut. Next, we observe the behavior of the algorithm over the course of the simulation. Finally, we compare GF-Part's performance with the state-of-the-art in terms of edge-cut.

4.4.1 Testbed

We implemented GF-Part in Repast HPC, a complete ABM simulation toolkit. It implements the core agent-based modeling concepts and features of Repast Symphony (the current version of Repast for serial simulations) but is aimed at facilitating large-scale distributed ABM simulations [15]. More specifically, Repast HPC targets both models with a small number of highly complex agents as well as models with a large number of relatively simple agents. It was built in C++ but provides interfaces for both C++ and ReLogo ABM implementations [72]. The toolkit does not provide specific mechanisms for graph partitioning and load balancing. It supports the notion of community; how-

ever it has no community-detection functionality. If a network of agents is deliberately distributed by the user on processes according to community structure, Repast HPC then allows the user to maintain and track the communities.

We ran the Repast HPC experiments on a CPU-cluster with the following configuration: 3,744 compute cores (Intel Xeon 64-bit processors), 240TB of configured storage, and a 56Gb Infiniband interconnect.

4.4.2 Metrics

The performance of a graph-partitioning algorithm can be measured by a number of metrics [73]. In addition to *edge-cut*, which is the metric we use in the objective function of the problem (see Section 4.2.1), we measure the number of vertices being moved during the redistribution phase (*migrations*).

4.4.3 Datasets

We used two different benchmark toolkits to generate synthetic graphs: LFR and SNAP's KronGen and KronFit tools. The properties of these graphs are listed in Table 4.1.

Table 4.1: Datasets

Dataset	$\|V\|$	$\|E\|$	Type
Kron1M	1,048,576	7,054,286	Synth.
KronDolphin	65,536	5,338,764	Extrapol.
LFR-1M-mu0.1	1,048,576	10,537,206	Synth.
LFR-1M-mu0.3	1,048,576	10,537,206	Synth.

4.4.3.1 *Kronecker Network Generation Model*

One of the graph generators we used was SNAP's Kronecker network generation model that was previously introduced in Chapter 3. The model is empirically proven to effectively model the structure of real networks and is mathematically tractable [62].

We used the Kronecker generator, KronGen, to extrapolate a large version of a small real network. Specifically, we used KronFit to extract the essential features of the *dolphins social network* graph [65]. We then used these features to grow the graph with KronGen (see Table 4.1). In addition, we used KronGen to generate a synthetic graph of over one million vertices. The characteristics of the graph are very similar to those of numerous real-world networks.

4.4.3.2 *Lancichinetti–Fortunato–Radicchi Benchmark Tool*

We used the (LFR) benchmark tool [66], previously introduced in Chapter 3, to generate two synthetic graphs with realistic characteristics. The tool generates networks with power-law distributions. It allows the user to control parameters such as the mixing parameter (the average percentage of external edges in the degree of a node), average node degree, and community size.

4.4.4 *The Impact of the Selection Strategy*

In this section, we study the effects on the edge-cut of several selection strategies. The strategies of choice were introduced in Section 4.3.3 and are labeled here as BI (best improvement), FI (first improvement), BI+R (best improvement with randomness), FI+R

(first improvement with randomness). In this set of experiments, FLS was called ten times by GLS.

As the results in Table 4.2 demonstrate, all four selection strategies improve the initial edge-cut, the product of the initial random partitioning. As expected, BI yields the best edge-cut; however, it does not appear to benefit from randomization. On the other hand, FI+R offers the worst performance in almost all cases. This result is due to the lack of more thorough exploration of the local neighborhood and the presence of opportunities to jump to a random place in the global search space. This search strategy works best in the case of KronDolphins. The highly dense graph does not lend itself well to exhaustive search strategies. FI also offers improvement over the initial edge-cut. In addition, it performs slightly better than FI+R in most cases, because the search does not undergo as much diversification. However, the degree of intensification is not as high as in BI, which could explain the inferior performance with respect to edge-cut.

4.4.5 The Effect of λ on the Edge-cut

The λ parameter is involved in the modification of the objective function. It controls how much the penalties take effect and modify the objective. In this experiment, we study how much that augmentation affects the edge-cut, if at all.

Table 4.2: The Impact of Label Selection on the Edge-cut (Kron1M)

Dataset	Initial	BI	FI	BI+R	FI+R
Kron1M	5,239,032	4,189,813	4,196,089	4,189,813	4,196,583
KronDolphin	2,594,692	2,505,343	2,505,242	2,505,349	2,505,229
LFR-1M-mu0.1	10,217,807	9,454,239	9,476,337	9,454,672	9,474,597
LFR-1M-mu0.3	10,315,637	9,756,285	9,770,775	9,756,315	9,770,152

Table 4.3: The Effect of λ on the Edge-cut

Dataset	$\lambda = 0$	$\lambda = 0.05$	$\lambda = 0.5$	$\lambda = 1$	$\lambda = 5$
Kron1M	4,208,061	4,189,821	4,202,185	4,205,245	4,269,483
KronDolphin	2,505,562	2,505,065	2,505,020	2,505,563	2,528,861

Table 4.4: Performance for Kron1M (BI)

Iterations	Edge-cut	Migrations
0	5,239,032	0
1	4,209,685	16,025
5	4,189,818	16,041
10	4,189,822	16,041
20	4,189,821	16,041
50	4,189,815	16,041

Table 4.5: Performance for KronDolphin (BI)

Iterations	Edge-cut	Migrations
0	2,594,692	0
1	2,505,568	8,055
5	2,505,068	8,056
10	2,505,060	8,056
20	2,505,041	8,056
50	2,505,022	8,056

The results listed in Table 4.3 show that λ has some effect on the edge-cut. Increasing its value allows the algorithm to get out of a local optimum faster. However, after a certain point, $\lambda = 0.5$, the edge-cut values begin to increase. This is valid for the Kron1M dataset. KronDolphin, on the other hand, seems to benefit from higher values of the parameter. As a result of the graph's high density and lack of well-structured clusters of vertices, the search benefits from randomness. The higher λ is, the closer the search compares to randomness. Generally, the algorithm is tolerant to the value of λ, but the parameter can be used to aid the search strategy [69].

4.4.6 Evolution of Edge-cut and Migration Over Time

In this section, we study the evolution of the algorithm over time. The GLS procedure in GF-Part calls LFS in a loop, which dictates how many times a percentage of the vertices is re-activated to perform a local search again. We observe the effect of this variation on

the values of the edge-cut and the number of migrations. We present results for Kron1M and KronDolphin, with the BI selection strategy, in Tables 4.4 and 4.5, respectively. In both cases, the results show that the edge-cut continues to improve with the increase in run time. However, the number of vertices to be moved levels off very quickly. This could be attributed to the fact that the algorithm is sequential on each processor (and BI is deterministic). Therefore, whenever a vertex undergoes a change in label, it affects a similar number of agents every time during an iteration of FLS. This result is likely to change if we run GF-Part in a truly distributed environment and allow a degree of randomness in the label selection strategy.

4.4.7 Comparison with State-of-the-Art

In this section, we compare the performance of GF-Part with that of ParMetis [48, 67] in terms of both edge-cut and execution time for all graphs. The results are shown in Table 4.6. In some cases, GF-Part produces a better result; in others, ParMetis does. In the case if the LFR graphs, which are composed of numerous small communities of vertices, GF-Part with BI does not seem to be able to get out of local optima. It is worth exploring the effects of modifying the costs of the features and introducing randomization in the label selection strategy.

Table 4.6: Comparison between GF-Part (GF-P) and ParMetis(PM)

Dataset	GF-P Edge-cut	PM Edge-cut	GF-P Time (s)	PM Time (s)
Kron1M	**4,209,677**	4,619,657	57.02	49.48
KronDolphin	**2,505,020**	2,549,599	**21.96**	26.21
LFR-1M-mu0.1	9,550,822	1,630,293	151.39	69.67
LFR-1M-mu0.3	9,817,406	4,486,306	189.96	79.21

It is worth noting that a significant part of the execution time of FG-Part is spent on synchronization, as a result of the inefficient implementation of these routines in Repast HPC. In addition, the results were obtained with the BI local search strategy, which is exhaustive and therefore slows down the convergence.

4.5 Conclusion

In this chapter, we presented a metaheuristic solution to the graph partitioning problem. To the best of our knowledge, we are the first to apply the combination of Guided Local Search and Fast Local Search to address the graph partitioning problem. We demonstrated that the algorithm is characterized by high locality which suits the nature of the problem. Our algorithm allows the search for a solution to occur from the perspective of each vertex. Therefore, GF-Part is well suited for partitioning the underlying graphs of agent-based model simulations.

In addition, GF-Part allows one to tailor the partitioning process to a specific graph by involving problem-specific features in the process. The features, their costs, and utility values can play a critical role in the effectiveness of the partitioner. As the underlying graphs of modeled complex systems can differ according to the nature of the system, the problem-specific parameters in GF-Part allow the algorithm to tune itself and reflect any distinctive characteristics pertinent to the partitioning process.

In the next chapter, we focus on the role of graph partitioners in the simulations of complex systems in distributed agent-based environments. We design a study that evaluates the effects of different types of graph partitioning algorithms on the performance of several of the state-of-the-art agent-based modeling platforms.

CHAPTER 5: NETWORK PARTITIONERS IN DISTRIBUTED AGENT-BASED MODEL SIMULATIONS (DESIGN OF A PERFORMANCE STUDY)

5.1 Overview

Most distributed ABM simulation platforms do not have partitioning and load-balancing mechanisms. However, most of them attempt to reduce the inter-processor communication via alternative, typically programmatic, approaches such as duplicate agents, delayed synchronization, bulk communication, coalesced messaging, selective agent simulation (i.e., simulating only a subset of the population per iteration with the assumption that not all agents are active at the same time). The few ABM platforms that consider some sort of partitioning, do so on the basis of standard geographic divisions such as zip code or city lines. This strategy, combined with a lightweight, skinny agent design and selective agent simulation, is usually enough to yield a significant reduction of inter-processor communication. However, such restrictions might take away from the utility of ABMs in representing an interaction network of agents with heterogeneous features and behaviors. While smart object design from a programmatic stand-point is necessary, it should not diminish the freedom a user needs to create a useful and meaningful model. Furthermore, heavily relying on spatial structure can lead to computational irreducibility (the idea that some computations cannot be sped up by any shortcut) when certain kinds of behavioral rules are applied [18,74].

In this study, we evaluate the usefulness of partitioners in accelerating the simulations of ABMs. We focus on three families of de facto standard partitioners, which represent both traditional graph-partitioning methods and more recent distributed algorithms, de-

signed for large-scale partitioning. We conduct our performance study in the context of Repast HPC, an open source agent-based modeling environment, widely used by scientists from various domains. The goal of our study is to evaluate whether large-scale ABM simulations benefit from the use of network-partitioning methods. Further, we propose to investigate when and how distributed, topology-aware partitioners perform better than the traditional multilevel graph-partitioning methods, which assume global knowledge and access to the network.

5.2 The Graph-Partitioning Problem

As we stated in Section 2.4, graph partitioning is a well studied problem. The challenge of its complexity has led to the development of numerous approximation and heuristic-based algorithms. We focus on *balanced k-way graph partitioning* for distributed agent-based simulations, because of its high relevance and applicability to distributed computing. This problem concerns the partitioning of the graph into equal-sized components. In many techniques, the equality requirement is relaxed by allowing it to vary by a small factor. This factor is usually included in a cost function that evaluates the efficiency of the algorithm.

5.2.1 Preliminaries

Let $G = (V, E)$ be an undirected, unweighted graph with a vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and an edge set $E = \{e_1, e_2, \ldots, e_m\} \subseteq (V \times V)$, representing a set of agents and the interactions among them, respectively. In addition, let $N(v_i) = \{v_j \in V : \langle v_i, v_j \rangle \in E\}$ denote the neighborhood set of vertex v_i, and let $d(v_i)$ denote the degree of v_i, that is

$d(v_i) = |N(v_i)|$.

5.2.2 Problem Statement

Given a graph $G = (V, E)$, partition V into k disjoint parts, $\Pi = \{\pi_1, \pi_2, \ldots, \pi_k\}$, such that:

$$(1 - \epsilon_l)\frac{|V|}{k} \leq |V(\pi_i)| \leq (1 + \epsilon_u)\frac{|V|}{k}, \tag{5.1}$$

while minimizing the size of the global edge cut K, defined as $K = \sum_{i<j} E_{ij}$, where $E_{ij} = \{\{u, v\} \in E : u \in \pi_i, v \in \pi_j\}\}$. In addition, ϵ_l and ϵ_u denote the lower and upper vertex imbalance parameters.

5.3 Standard Partitioning Methods

In this section, we present some of the most widely used graph-partitioning methods of recent years. They have gained wide acceptance in the scientific world and are often used as the de facto standards in graph partitioning for comparison purposes.

5.3.1 METIS

METIS is a software package used for serial partitioning of graphs and meshes, as well as for producing fill reducing orderings for sparse matrices [75]. The algorithms available in METIS are based on the multilevel graph-partitioning paradigm. This paradigm consists of three phases: graph coarsening, initial partitioning, and uncoarsening. The graph-coarsening phase produces a series of successively smaller graphs by collapsing together pairs of adjacent vertices. This process reduces the original graph to a new one – of only a

few hundred vertices. In the initial partitioning phase, the coarsened graph is partitioned using relatively simple algorithms such as that by Kernighan-Lin [76]. This phase is quick because of the small size of the graph. The final phase, uncoarsening, takes the partitioning and projects it to successively larger graphs derived from the coarsened graph. This is accomplished by assigning pairs of vertices, previously collapsed together, to the same partition, to which their collapsed common vertex belonged. The uncorsening phase incorporates various heuristics to refine the partitions after each projection step. This phase ends when the partition has been projected all the way back to the original graph.

5.3.2 ParMETIS

ParMETIS is an MPI-based library that extends METIS and offers parallelized versions of the functions in the original tool [77]. Specifically, the parallel graph-partitioning algorithm used in ParMETIS is the parallelized version of the serial *multilevel k-way partitioning* algorithm used in METIS [48, 67].

5.3.3 KaFFPa

KaFFPa (Karlsruhe Fast Flow Partitioner) is a multilevel graph-partitioning algorithm in the Karlsruhe High Quality Partitioning (KaHiP) suite [78]. The algorithm uses local improvement algorithms based on max-flow and min-cut computations and more localized FM (a heuristic-based network-partitioning algorithm by Fiduccia and Mattheyses) searches. It also relies on global search strategies borrowed from multi-grid linear solvers.

Another noteworthy technique for graph partitioning in KaHiP is KaFFPaE, a distributed evolutionary algorithm. It uses KaFFPa as a base to provide effective crossover and mu-

tation operators for maintaining high population diversity. KaFFPaE's latest features include a local improvement scheme for graph partitions that allows strict balance constraints. By combining local searches, which individually violate the balance constraint into a more global feasible improvement, with a balancing algorithm, KaFFPaE is able to guarantee any balance constraint [52].

5.3.4 Ja-Be-Ja

Ja-Be-Ja is a fully distributed algorithm that uses local search and simulated annealing techniques for two types of graph partitioning: edge-cut and vertex-cut [79]. The algorithm is massively parallel – designed to partition extremely large graphs. The algorithm is claimed to achieve such scalability through its data locality, simplicity, and lack of synchronization requirements.

The basic idea is to assign colors to nodes uniformly at random and then apply a local search to push the configuration towards lower energy states (min-cut). Each vertex iteratively selects vertices either among its neighbors or via a random sample, and investigates the pair-wise utility of a color exchange. In order to preserve the size of the partitions, the colors do not change independently, but are swapped, instead. To ensure that the algorithm does not get stuck in a local optimum during a local search, Ja-Be-Ja uses simulated annealing. The vertices of the graph are processed periodically and asynchronously such that each vertex only has access to the states of its neighbors and a small set of random vertices in the graph.

5.4 Experimental Setup

In this section, we describe the testbed used in our experimental study.

5.4.1 Modeling Environment

As in previous chapters, we turn to Repast HPC as our modeling environment. Recall that the toolkit does not provide graph partitioning and load balancing functionality. It supports the notion of community; however, it relies on the user to identify the communities. If a network of agents is deliberately distributed by the user on processes according to community structure, Repast HPC then allows the user to maintain and track the communities.

5.4.2 Benchmarks

As Barabási points out, despite the rich diversity of complex networks, they all share common structures and evolution as a result of being governed by the same organizing principles [80]. Consequently, we can categorize (see Section 2.1 for categories) and analyze them via the use of a common set of mathematical and computational tools.

Although ABMs are an abstraction, they have the potential to represent reality at a high degree of detail and even simulate real-world phenomena in real time. The fidelity of an agent-based model relies in part on the credibility of the underlying contact network [81]. In that regard, fusing data from different real-world sources is expected to enhance the credibility of a model. Potential data sources include censuses, data from a telecommunication service provider (cellular records) and person-person contact data based on logs of

Bluetooth connectivity between devices. Such fused data sets present the most effective robustness and scalability tests for ABM simulation environments; however, they are not yet readily available.

Here as in other chapters of the dissertation, we use synthetic networks with realistic topological characteristics. We use the Kronecker network-generation model, which is empirically proven to effectively model the structure of real networks and is mathematically tractable [62]. The generator has been proven to obey not only all the main static network patterns that have appeared in the literature, but also temporal evolution patterns [63]. Two essential network characteristics are heavy-tailed degree distributions and small diameters. Heavy-tailed degree distributions, following a power law, describe real-world networks that contain numerous nodes with very small degrees and a small number of nodes (aka "hubs") with very high degrees [80]. In contrast, Erdős-Rényi random networks are characterized by a Poisson degree distribution. Most current generators focus on only one or two patterns, and neglect the others. For example, many models involve some form of preferential attachment, which yields networks with power-low tails [27,82]. Others, like the small-world generator, obey the small-diameter pattern [25]. In contrast, the Kronecker network generator has been rigorously proven to address all known characteristics.

In addition, the generator includes KronFit, a linear time, scalable algorithm for fitting the Kronecker graph generation model to real networks [64]. One of the benefits of this feature is that it allows us to generate extrapolations of smaller real networks. These extrapolations are larger versions of the original graphs, evolved via a set of temporal growth patterns.

We are using the Kronecker generator to extrapolate a large version of a small real net-

work. The benchmark set includes several networks with large sizes (with millions of nodes) and varied parameters (such as community size and community interconnectivity, etc). Each network is dynamic in nature – with a fixed number of mobile nodes (agents). To further ensure high fidelity of our models, we vary the type and complexity of the modeled phenomena (see Section 5.4.4).

5.4.3 Methods

We compare the performance of the following de facto standard graph-partitioning methods:

- METIS and ParMETIS: libraries accessed in an ABM implementation through a series of function calls.
- KaHIP: a C/C++ library, whose algorithms can be accessed in the ABM implementation through a set of function calls.
- Ja-Be-Ja: No suitable implementation is readily available for this algorithm; therefore, a strategy is to implement it in Repast HPC.
- Random: The agent population is equally and randomly distributed across the processors.

5.4.4 Simulation Model

Our model of choice is similar to the one we described in Section 3.2.1, but we include it here for completeness and to allow additional clarification pertaining to this study. The SIR ABM simulates the spread of disease in a population of agents such that each agent

falls in one of three categories: susceptible, infected, or recovered. We simulate the spread of disease through the exchange of messages between connected agents. Each message carries the health condition of the sender as payload. The disease propagation process is governed by two parameters, i.e., the rate of infection and the rate of recovery. We assume that the population size remains constant; that is, we do not include an agent birth and death mechanism. Each agent updates its status depending on its own health condition and that of its neighbors. We assume that each agent sends (receives) a message, symbolizing social contact and disease transmission, to (from) all of its neighbors at each iteration of the simulation. The stopping condition of the simulation is the complete eradication of the disease, i.e., when the number of infectious agents reaches zero, or a predetermined number of simulation steps.

The recovery process of each agent is stochastic in nature. Specifically, for each infected agent the algorithm draws a random number from a uniform distribution between 0 and 100. If the number is less than the recovery rate, the agent's state changes to 'recovered' (and therefore, immune). If not, the agent remains infected. Similarly, for all susceptible agents, the algorithm draws a random number from a uniform distribution between 0 and 100. If the number is less than the rate of infection, the agent's state changes to 'infected'.

The dynamics of the ABM directly affect the performance of the simulation. To investigate how the overhead due to communication and computation is affected, we vary the complexity of the model. The following subsections provide the specifics.

5.4.5 Varying Social Contact

In our simulations, an agent's neighborhood represents the set of all regular contacts of that agent. However, not all of these contacts have to be active every simulation step. We

can vary the number of active links per iteration from none to all. In addition, we allow for chance encounters with strangers. The frequency of this type of contacts will also be varied. Varying the amount and patterns of contact will allow us to study the benefits of network partitioning in different communication scenarios.

5.4.6 Varying Mobility

We vary the fraction of the population that completely changes location, and consequently community membership, every iteration. Varying the mobility parameter allows us to study the trade-off between the benefits of good partitioning and the cost of detecting the partitions and redistributing the network.

5.4.7 Varying Computational Workload

To investigate the effect of the processor workload on the simulation performance, we let agents compute a "Fast Fourier Transform (FFT)" [83]. We vary the workload by using different sizes of input for the FFT calculus.

5.5 Experimental Study

The purpose of this study is to determine whether graph partitioning is beneficial in distributed ABM simulations. The hypothesis is that there will be a discernible and even maybe a significant benefit. However, we would like to find out whether there are any trade-offs and how they fluctuate with variations in the network topology and characteristics of the modeled phenomena. To gain clarity on these issues, we propose the follow-

ing sets of experiments:

- This experiment is intended to show how the ABM's execution time varies with the amount and frequency of social contact. One of the goals is to also determine the effect of chance communication with strangers. This type of contact is expected to contribute primarily to the amount of inter-processor communication.

- This experiment is designed to demonstrate how the mobility parameter affects the execution time. One can plot the performance-deterioration trend over time (using execution time per iteration) for different values of the mobility parameter.

- This experiment is intended to investigate the effects of the computational load per node in different work distribution scenarios. Some possibilities are: 1) nodes work only if they are socially active during a given iteration; 2) all nodes work every iteration; 3) a random subset of nodes works every iteration; etc. One can then plot the relationship between total execution time and amount of workload for all partitioning schemes. Furthermore, this study illustrates how the imbalance of the processors' workloads deteriorates over several iterations, between network repartition procedure calls.

- Scaling is an important property of any parallel and distributed simulation. The next two experiments focus on how the ABM scales as we vary two different parameters. The first one demonstrates how the performance scales up as we double the population and number of processors repeatedly (weak scaling). The second one is designed to show how the execution time of a fixed simulation changes as the available computing power repeatedly doubles (strong scaling).

5.6 Conclusion

In this chapter, we proposed a study on the effects of graph partitioning on different agent-based modeling environments. Its purpose is twofold. First, it aims to demonstrate that agent-based modeling environments improve in performance as a result of incorporating graph partitioning algorithms. Second, it is designed to determine whether topology-aware graph partitioning approaches are a better fit for distributed simulations of complex systems than traditional graph partitioners. In the next chapter, we present our vision for future work. Specifically, we focus on applying Catastrophe Theory concepts to facilitate more effective remapping of networks in distributed ABM simulations.

CHAPTER 6: APPLYING CATASTROPHE THEORY TO NETWORK REMAPPING IN DISTRIBUTED AGENT-BASED MODEL SIMULATIONS (FUTURE WORK)

6.1 Overview

As demonstrated in Chapter 3, applying community detection as an integrated process in the dynamics of a distributed ABM leads to a significant drop in the inter-processor communication overhead. However, our study also demonstrated that a distributed ABM simulation could suffer from additional overhead associated with the periodic redistribution of the model. Essentially, the benefit of exploiting locality is oftentimes offset by overhead due to inter-processor agent migration. In order to properly take advantage of locality in the simulation of ABMs, we have to address the communication overhead due to redistribution and synchronization of the model. With this in mind, we turn to Dynamical Systems Theory and attempt to characterize the processes creating the mentioned overhead.

As with the model-partitioning methods presented in Chapters 3 and 4, the ideas here attempt to exploit the intrinsic relationship between the structural properties of the underlying ABM network and the dynamics of its elements. As we have noted previously, one of the main characteristics of complex systems is the emergent global behavior that is a spontaneous outcome of the behaviors and interactions of the elements on a microscopic scale. These dynamic processes give rise to a complex topology and often heterogeneous structure. Knowing and being able to predict how this structure changes over time can inform the design of more efficient model-partitioning techniques in the context of distributed computing. Unfortunately, decomposing the system to its most basic

autonomous elements does not provide insight into the link between its global characteristics and local individual behaviors. This observation is due to the very nature of the self-organizing principles governing the system dynamics – they arise from the collective and unsupervised dynamics of the multitude of autonomous elements [20].

On the other hand, there is compelling evidence that fluctuations and complications in the system are visible at different scales [20]. Therefore, we can expect that changes in the system on a microscopic level could propagate and affect the system at a global scale. In many complex networks correlations among the constituent elements are typical and therefore may generate cascades of microscopic events disrupting the system at all levels. Barrat et al. note that multilevel complications are statistically encoded in the heavy-tail distributions characterizing the structural properties of most real-world networks [20,26]. The visibility of fluctuations at different resolutions of the system allows us to observe them at a more global scale and to discern patterns that could be useful in predicting the behavior of the system.

As a result of the ubiquity of complex systems in various domains – all sharing characteristics such as emergent behavior, heterogeneous structure, scale-free properties, and non-trivial statistical correlations – it is intuitive to speculate that there may be certain mechanisms, governing the system dynamics, that are common across systems with otherwise disparate behaviors. Recent large-scale data analyses demonstrate that there are, indeed, common self-organizing principles that govern the emergent phenomena in otherwise completely different systems. These observations are very much in agreement with the notion of universality addressed in statistical physics in relation to "phase transitions" in dynamic systems. More specifically, universality refers to the fact that very different complex systems demonstrate large-scale properties that were derived by the same set of statistical laws [20]. These systems are very different on a microscopic level

but similar on a global scale. Universality does not imply equivalence; instead, it attempts to identify any dynamic mechanisms responsible for generating these similarities regardless of the microscopic details of the system.

Studies of the dynamics of complex systems conducted in the last two decades have provided compelling evidence that such systems might exhibit critical transitions, also known as tipping points, that trigger dramatic shifts in the systems' states. Some examples of such dramatic changes in real systems include the collapse of financial markets, the sudden disappearance of natural species, or epileptic seizures [34, 84]. Such systemic shifts could not only pose challenges to maintaining stability, but also prove catastrophic. Therefore, it would be beneficial to devise methods for early detection of such dramatic events. Until recently, it was considered almost impossible to predict sudden systemic shifts, mainly because changes in the systems leading up to the transitions usually are either not detectable or seemingly unrelated [84]. In addition, most models of complex systems are not accurate enough and therefore inadequate in capturing the subtleties of such seemingly sudden events.

Surprisingly, recent studies demonstrate not only that complex systems exhibit symptoms of imminent critical transitions but also that these warning signs appear common across different phenomena regardless of the disparate system dynamics. The branch of dynamics known as Catastrophe Theory is concerned with systems that, under particular conditions, demonstrate sudden shifts from one form to another. These sudden jumps are associated with a number of catastrophe flags. Some of the leading indicators of critical transitions that may occur in non-equilibrium dynamics include [84]:

- Critical Slowing Down. This indicator is considered essential in capturing tipping points in a wide range of natural systems such as cell signaling pathways and cli-

mate. The idea behind this warning signal is that complex systems become increasingly slow to recover from perturbations as they approach a critical transition. This phenomenon occurs in continuous models approaching fold bifurcation [84].

- Fluctuations (Flickering). Persisting small system fluctuations can increase in intensity thereby signaling that a system is entering an unstable regime, a precursor to a transition to another, perhaps contrasting, state [85].

- Spatial Patterns. As stated in Chapter 2.2.3, some structural patterns emerge when the behaviors of individual entities are overpowered by the influence of their neighbors. Such shifts in behavior may generate specific patterns indicative of an imminent critical transition.

We argue that the dynamic instability of a distributed ABM simulation can be considered a catastrophic behavior; a small shift in the workload distribution or sudden large bursts of inter-processor communication can cause systemic failure or at least prohibitively large simulation delays. In addition, the distributed simulations of ABMs could be represented as self-organizing dynamic networks and therefore may be suitable candidates for the application of Catastrophe Theory.

6.2 Preliminaries

We represent the agent communities or partitions, detected as described in Chapters 3 and 4, as vertices in a graph. The edges in this graph denote the communication links between the communities. It is safe to conclude that the network portrayed by this graph is complex in nature – it consists of a large number of nodes (denoting the communities in an even larger network), whose attributes and interactions evolve over time. In par-

ticular, a suitable attribute of a node in this network is the changing number of agents that belong to a community. Furthermore, instead of portraying the potential multitude of communication links between pairs of nodes as multi-edges, we can consolidate them into a single weighted edge per pair and present the number of links as the edge weight. Thus, the dynamics of the newly formed complex network should reflect the dynamics of the inter-process communication overhead in the original distributed ABM simulation.

6.3 Assumptions

We base our proposal to apply Catastrophe Theory to the problem of network redistribution on the following assumptions:

- Simulated ABMs are partitioned on distributed platforms using integrated label propagation community detection. This approach allows us to avoid the overhead of the periodic application of any commonly used graph-partitioning algorithm.
- We assume that the underlying network of the ABM is dynamic and therefore changes structurally over time. These changes progressively reduce the benefits of the community-to-processor mapping by increasing the inter-processor communication overhead. This trend eventually may disrupt the flow of the simulation and even render the system unusable.
- The underlying ABM network is massive – it contains millions of agents and possibly billions of connections between them. Such test networks provide for a more realistic testing environment. Moreover, the size of the network exacerbates the overhead due to synchronization and load-balancing. Thus, we can test our solutions of these issues more effectively.

6.4 Scope of Proposed Future Research

The envisioned long term goal of this research direction is to demonstrate the applicability of Catastrophe Theory to the dynamic behavior of distributed simulations of massive agent-based models (or dynamic networks in general). The purpose of the proposed approach is to lay the groundwork for this research by focusing on the actual response strategies triggered by potential catastrophe warning signals. With these response mechanisms in place, we will be able to focus on the intricacies of applying Catastrophe Theory in distributed computing simulations at a later stage of this research.

For the first stage of the study, we propose focusing on the gradual synchronization and re-balancing of the system using staged artificial catastrophe-prediction signals. The behavior of these indicators is stochastic in nature and thus any decision-making procedures that they trigger will also be probabilistic. Specifically, we envision that the "strength" of these signals will indicate how many of the agents due for migration will be redistributed to different processors, as well as which communities might have to be moved in order to re-balance the processor loads. In this stage, we only focus on devising the methods for gradual synchronization (i.e., agent migration after change in the community structure of the ABM network) and processor load-balancing. We propose the use of artificial critical transition (catastrophe) flags to indicate the potential occurrence of critical system state shifts. These flags can guide the synchronization and load-balancing processes to avoid systemic failures or simulation delays.

CHAPTER 7: CONCLUSION

Agent-Based Modeling has been gaining momentum as the preferred simulation approach with respect to complex systems. ABMs can adequately address the complex and decentralized nature of such systems as they are comprised of numerous autonomous, possibly heterogeneous, interacting agents. This modeling paradigm facilitates the representation of complex systems at different levels of granularity – from the point of view of the constituent elements at microscopic level, through their organization into groups at mesoscopic level, to the complex global phenomenon at macroscopic level. ABMs allow the model to mimic the structure of the original system as it is built bottom up. The elements at microscopic level are modeled as a network of interacting agents.

As the global behavior in many complex systems emerges at relevant scales, oftentimes the simulations must be run on high performance computing platforms. The performance of parallel simulations of large agent-based models distributed across multiple CPUs is strongly dependent on the distribution of the agents among the processors. The objective of this dissertation is to address the high inter-process communication overhead in distributed simulations of massive self-organizing agent-based models.

To address the communication bottleneck, we proposed two different partitioners that exploit certain topological characteristics of the ABM network. One of the key properties of many complex networks is community structure – the inherent organization of the network into groups of densely connected elements that are loosely connected with other such dense groups. Thus, in the first part of the dissertation we observed that community structure in self-organizing complex networks could be loosely associated with the communication overhead incurred in distributed simulations of such networks. We proposed

a network partitioner based on label-based community detection that performs seamless continuous community detection as part of the dynamics of the simulated ABM.

While, the LPA-based algorithm seamlessly integrates with the dynamics of modeled systems to partition them without incurring additional costs, it runs the risk of getting stuck in a local optimum and not achieving an optimal global partition. To remedy this drawback of our fist algorithm, in the second part of this dissertation we proposed GF-Part. It uses a combination of the Guided Local Search and Fast Local Search metaheuristics to guide the partitioning process to a better global solution. We demonstrated that the algorithm fits the decentralized nature of complex systems well. It successfully reduces the search space to speed up the partitioning process. In addition, it can use attributes of the specific system being modeled to facilitate the process of avoiding local optima. The approach allows the use of problem-specific properties to be used in the partitioning process thereby navigating the search space in a more effective and efficient manner.

In this dissertation, we demonstrated that both of our partitioners have strong locality. When applied to graphs with high modularity, they are able to effectively partition the graphs on distributed high performance computing platforms. As future work we intend to study the effects of different types of graph partitioning algorithms on synthetic and real-life graphs modeled by state-of-the-art ABM platforms. The goal is to determine not only to what extent such platforms benefit from graph partitioning, but also what types of algorithms are most suitable.

In addition, we plan to use concepts from Dynamical Systems Theory to capture critical changes in the dynamics of complex systems to facilitate more efficient simulations. Specifically, we intend to apply Catastrophe Theory in an effort to detect when an ABM network distributed on a HPC platform has to be redistributed in order to avoid a com-

munication bottleneck.

LIST OF REFERENCES

[1] G. Vigueras, M. Lozano, and J. M. Orduña, "Workload balancing in distributed crowd simulations: The partitioning method," *The Journal of Supercomputing*, vol. 58, no. 2, pp. 261–269, 2011.

[2] Y. Wang, M. Lees, W. Cai, S. Zhou, and M. Y. H. Low, "Cluster based partitioning for agent-based crowd simulations," in *Winter Simulation Conference*. Winter Simulation Conference, 2009, pp. 1047–1058.

[3] B. Zhou and S. Zhou, "Parallel simulation of group behaviors," in *Proceedings of the 36th Conference on Winter simulation*. Winter Simulation Conference, 2004, pp. 364–370.

[4] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," vol. 21, no. 4. ACM, 1987, pp. 25–34.

[5] J. J. Corner and G. B. Lamont, "Parallel simulation of uav swarm scenarios," in *Proceedings of the 36th Conference on Winter Simulation*. Winter Simulation Conference, 2004, pp. 355–363.

[6] C. M. Macal and M. J. North, "Agent-based modeling and simulation: Abms examples," in *Winter Simulation Conference*, 2008, pp. 101–112.

[7] D. A. Luke and K. A. Stamatakis, "Systems science methods in public health: dynamics, networks, and agents," *Annual Review of Public Health*, vol. 33, pp. 357–376, 2012.

[8] F. Hinkelmann, D. Murrugarra, A. S. Jarrah, and R. Laubenbacher, "A mathematical framework for agent based models of complex biological networks," *Bulletin of Mathematical Biology*, vol. 73, no. 7, pp. 1583–1602, 2011.

[9] S. L. Mabrya and L. F. Bicb, "Bridging semantic gaps with migrating agents," *Network*, vol. 1, no. P2, p. P3, 1999.

[10] C. M. Macal and M. J. North, "Introduction to agent-based modeling and simulation," in *MCS LANS Informal Seminar*, 2006.

[11] ——, "Tutorial on agent-based modelling and simulation," *Journal of Simulation*, vol. 4, no. 3, pp. 151–162, 2010.

[12] S. Swarup, S. G. Eubank, and M. V. Marathe, "Computational epidemiology as a challenge domain for multiagent systems," in *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 1173–1176.

[13] M. J. North, N. T. Collier, and J. R. Vos, "Experiences creating three implementations of the repast agent modeling toolkit," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 16, no. 1, pp. 1–25, 2006.

[14] U. Wilensky, "Netlogo," Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, http://ccl.northwestern.edu/netlogo/, 1999.

[15] N. Collier and M. North, "Repast HPC: A platform for large-scale agent-based modeling," *Large-Scale Computing Techniques for Complex System Simulations*, pp. 81–110, 2011.

[16] G. Cordasco, F. Milone, C. Spagnuolo, and L. Vicidomini, "Exploiting D-Mason on parallel platforms: A novel communication strategy," in *European Conference on Parallel Processing*. Springer, 2014, pp. 407–417.

[17] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[18] H. Sayama, *Introduction to the modeling and analysis of complex systems*. Open SUNY Textbooks, 2015.

[19] W. Weaver, "Science and complexity," *American Scientist*, vol. 36, pp. 536–544, 1948.

[20] A. Barrat, M. Barthelemy, and A. Vespignani, *Dynamical processes on complex networks*. Cambridge University Press, 2008.

[21] Z. Toroczkai, "Complex networks," *Science-Based Prediction*, p. 94, 2005.

[22] R. Pastor-Satorras and A. Vespignani, "Complex networks: Patterns of complexity," *Nature Physics*, vol. 6, pp. 480–481, 2010.

[23] N. Corson, M. Aziz-Alaoui, R. Ghnemat, S. Balev, and C. Bertelle, "Modeling the dynamics of complex interaction systems: from morphogenesis to control," *International Journal of Bifurcation and Chaos*, vol. 22, no. 02, p. 1250025, 2012.

[24] P. Erdős and A. Rényi, "On random graphs I," *Publicationes Mathematicae Debrecen*, vol. 6, p. 290, 1959.

[25] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[26] C. Gros, *Complex and Adaptive Dynamical Systems: A Primer*. Springer International Publishing, 2015.

[27] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[28] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwang, "Complex networks: Structure and dynamics," *Physics Reports*, vol. 424, no. 4, pp. 175–308, 2006.

[29] J. D. Sterman, "Learning from evidence in a complex world," *American Journal of Public Health*, vol. 96, no. 3, pp. 505–514, 2006.

[30] G. An, Q. Mi, J. Dutta-Moscato, and Y. Vodovotz, "Agent-based models in translational systems biology," *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, vol. 1, no. 2, pp. 159–171, 2009.

[31] G. N. Gilbert, *Agent-based models*. Sage, 2008, no. 153.

[32] C. M. Macal and M. J. North, "Agent-based modeling and simulation," in *Winter Simulation Sonference*. Winter Simulation Conference, 2009, pp. 86–98.

[33] F. Castiglione, "Introduction to agent based modeling and simulation," in *Encyclopedia of Complexity and Systems Science*. Springer, 2009, pp. 197–200.

[34] H. Moon and T.-C. Lu, "Network catastrophe: Self-organized patterns reveal both the instability and the structure of complex networks," *Scientific Reports*, vol. 5, 2015.

[35] D. J. Barnes and D. Chu, "Agent-based modeling," in *Introduction to Modeling for Biosciences*. Springer, 2010, pp. 15–77.

[36] D. Helbing, "Agent-based modeling," in *Social Self-Organization*, ser. Understanding Complex Systems. Springer Berlin Heidelberg, 2012, pp. 25–70.

[37] F. Klügl and A. L. Bazzan, "Agent-based modeling and simulation," *AI Magazine*, vol. 33, no. 3, p. 29, 2012.

[38] E. Amouroux, P. Taillandier, A. Drogoul *et al.*, "Complex environment representation in epidemiology ABM: application on H5N1 propagation," *Journal of Science and Technology*, pp. 13–25, 2010.

[39] B. Walter, "Virtual environment UAV swarm management using GPU calculated digital pheromones," Ph.D. dissertation, Citeseer, 2005.

[40] P. Riley, "SPADES: a system for parallel-agent, discrete-event simulation," *AI Magazine*, vol. 24, no. 2, p. 41, 2003.

[41] S. Luke, G. C. Balan, L. Panait, C. Cioffi-Revilla, and S. Paus, "MASON: A Java multi-agent simulation library," in *Proceedings of Agent 2003 Conference on Challenges in Social Simulation*, vol. 9, 2003, p. 9.

[42] R. K. Standish and R. Leow, "EcoLab: Agent based modeling for C++ programmers," *arXiv preprint cs/0401026*, 2004.

[43] M. Kiran, P. Richmond, M. Holcombe, L. S. Chin, D. Worth, and C. Greenough, "FLAME: simulating large populations of agents on parallel hardware architectures," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, 2010, pp. 1633–1636.

[44] C. Márquez, E. César, and J. Sorribes, "A load balancing schema for agent-based spmd applications," in *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Accepted*, 2013.

[45] E.-G. Talbi, *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009, vol. 74.

[46] B. Hendrickson and R. W. Leland, "A multi-level algorithm for partitioning graphs," *SC*, vol. 95, no. 28, 1995.

[47] G. Karypis and V. Kumar, "Parallel multilevel k-way partitioning scheme for irregular graphs, department of computer science tech. rep. 96-036," *University of Minnesota, Minneapolis, MN*, 1996.

[48] ——, "Multilevel k-way partitioning scheme for irregular graphs," *Journal of Parallel and Distributed Computing*, vol. 48, no. 1, pp. 96–129, 1998.

[49] G. Karypis, "METIS and ParMETIS," in *Encyclopedia of Parallel Computing*. Springer, 2011, pp. 1117–1124.

[50] C. Walshaw and M. Cross, "JOSTLE: parallel multilevel graph-partitioning software–an overview," *Mesh Partitioning Techniques and Domain Decomposition Techniques*, pp. 27–58, 2007.

[51] F. Pellegrini, "Distillating knowledge about Scotch," *Combinatorial Scientific Computing*, no. 09061, 2009.

[52] P. Sanders and C. Schulz, "Think locally, act globally: Highly balanced graph partitioning," in *Experimental Algorithms*. Springer, 2013, pp. 164–175.

[53] M. E. Newman, "The structure and function of complex networks," *SIAM Review*, vol. 45, no. 2, pp. 167–256, 2003.

[54] B. Hou and Y. Yao, "Commpar: A community-based model partitioning approach for large-scale networked social dynamics simulation," in *IEEE/ACM 14th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, 2010, pp. 7–13.

[55] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proceedings of the National Academy of Sciences*, vol. 105, no. 4, pp. 1118–1123, 2008.

[56] G. M. Slota, K. Madduri, and S. Rajamanickam, "PuLP: Scalable multi-objective multi-constraint partitioning for small-world networks," in *IEEE International Conference on Big Data (Big Data)*, 2014, pp. 481–490.

[57] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Physical Review E*, vol. 76, no. 3, p. 036106, 2007.

[58] M. E. Newman, "Spread of epidemic disease on networks," *Physical Review E*, vol. 66, no. 1, p. 016128, 2002.

[59] K. Kothapalli, S. V. Pemmaraju, and V. Sardeshmukh, "On the analysis of a label propagation algorithm for community detection," in *Distributed Computing and Networking*. Springer, 2013, pp. 255–269.

[60] N. Collier and M. North, "Parallel agent-based simulation with Repast for High Performance Computing," *Simulation*, vol. 89, no. 10, pp. 1215–1235, 2013.

[61] J. Leskovec and R. Sosič, "SNAP: A general-purpose network analysis and graph-mining library," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 8, no. 1, p. 1, 2016.

[62] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 985–1042, 2010.

[63] J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos, "Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication," in *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 2005, pp. 133–145.

[64] J. Leskovec and C. Faloutsos, "Scalable modeling of real graphs using kronecker multiplication," in *Proceedings of the 24th International Conference on Machine Learning*, ser. ICML'07, 2007, pp. 497–504.

[65] D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson, "The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations," *Behavioral Ecology and Sociobiology*, vol. 54, no. 4, pp. 396–405, Sep 2003.

[66] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Physical Review E*, vol. 78, no. 4, p. 046110, 2008.

[67] G. Karypis and V. Kumar, "A parallel algorithm for multilevel graph partitioning and sparse matrix ordering," *Journal of Parallel and Distributed Computing*, vol. 48, no. 1, pp. 71–95, 1998.

[68] C. Voudouris and E. Tsang, "Partial constraint satisfaction problems and guided local search," *Proc., Practical Application of Constraint Technology (PACT'96), London*, pp. 337–356, 1996.

[69] ——, "Guided local search and its application to the traveling salesman problem," *European Journal of Operational Research*, vol. 113, no. 2, pp. 469–499, 1999.

[70] F. Rothlauf, *Design of Modern Heuristics: Principles and Application*, 1st ed. Springer Publishing Company, Incorporated, 2011.

[71] M. Gendreau and J. Potvin, *Handbook of Metaheuristics*, ser. International Series in Operations Research & Management Science. Springer US, 2010.

[72] J. Ozik, N. T. Collier, J. T. Murphy, and M. J. North, "The ReLogo agent-based modeling language," in *Winter Simulation Conference*, 2013, pp. 1560–1568.

[73] J. Leskovec, K. J. Lang, and M. Mahoney, "Empirical comparison of algorithms for network community detection," in *Proceedings of the 19th International Conference on World Wide Web*, 2010, pp. 631–640.

[74] S. Wolfram, *A new kind of science*. Wolfram Media Champaign, 2002, vol. 5.

[75] G. Karypis and V. Kumar, "A fast and highly quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, p. 359—392, 1999.

[76] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, 1970.

[77] G. Karypis, K. Schloegel, and V. Kumar, "ParMetis: Parallel graph partitioning and sparse matrix ordering library," *Version 1.0, Dept. of Computer Science, University of Minnesota*, 1997.

[78] P. Sanders and C. Schulz, "High quality graph partitioning." *Graph Partitioning and Graph Clustering*, vol. 588, no. 1, 2012.

[79] F. Rahimian, A. H. Payberah, S. Girdzijauskas, M. Jelasity, and S. Haridi, "A distributed algorithm for large-scale graph partitioning," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 10, no. 2, p. 12, 2015.

[80] A.-L. Barabási, *Network Science*, 1st ed. Cambridge University Press, 2016.

[81] M. Laskowski, B. C. Demianyk, M. R. Friesen, R. D. McLeod, and S. N. Mukhi, "Improving agent based models and validation through data fusion," *Online Journal of Public Health Informatics*, vol. 3, no. 2, 2011.

[82] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of Modern Physics*, vol. 74, no. 1, p. 47, 2002.

[83] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

[84] M. Scheffer, J. Bascompte, W. A. Brock, V. Brovkin, S. R. Carpenter, V. Dakos, H. Held, E. H. Van Nes, M. Rietkerk, and G. Sugihara, "Early-warning signals for critical transitions," *Nature*, vol. 461, no. 7260, pp. 53–59, 2009.

[85] D. Helbing, *Social self-organization: Agent-based simulations and experiments to study emergent social behavior*. Springer, 2012.